NetSDK 编程指导手册



V3.5.1

前言

目的

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。

本文档详细描述了开发包中各个函数的功能、接口以及函数之间的调用关系，并提供了代码示例。

符号约定

在本文档中可能出现下列标志，它们所代表的含义如下。

符号	说明
 **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
V3.5.1	新增 2.15 APP 管理章节	2025.07
V3.5.0	修改优化硬盘获取信息接口和错误码输出接口	2025.02
V3.4.11	全文优化语言。	2023.02
V3.4.10	● 新增 NVR6 系等设备登录业务注意事项。 ● 接口函数定义、结构体定义和枚举定义章节转移到正文，修改部分内容格式。 ● 删除 fisheye 鱼眼矫正库。	2021.05
V3.4.9	● 删除 avnetsdk 依赖库信息。 ● 新增 StreamConvertor 依赖库。	2021.03
V3.4.8	修改登录设备和搜索设备接口函数。	2020.02
V3.4.7	修订。	2016.08
V3.4.4	创建。	2016.01

目录

前言 I
第 1 章 内容简介 1
1.1 概述 1
1.2 环境要求 1
第 2 章 主要功能 3
2.1 SDK 初始化 3
2.1.1 简介 3
2.1.2 接口总览 3
2.1.3 流程说明 3
2.1.4 示例代码 4
2.2 设备登录 7
2.2.1 简介 7
2.2.2 接口总览 8
2.2.3 流程说明 8
2.2.4 示例代码 9
2.3 实时预览 13
2.3.1 简介 13
2.3.2 接口总览 14
2.3.3 流程说明 14
2.3.4 示例代码 16
2.4 录像回放 28
2.4.1 简介 28
2.4.2 接口总览 28
2.4.3 流程说明 28
2.4.4 示例代码 32
2.5 录像下载 47
2.5.1 简介 47
2.5.2 接口总览 47
2.5.3 流程说明 48
2.5.4 示例代码 51
2.6 实时预览转码 66
2.6.1 简介 66
2.6.2 接口总览 66
2.6.3 流程说明 67
2.6.4 示例代码 67
2.7 录像回放转码 73
2.7.1 简介 73
2.7.2 接口总览 73
2.7.3 流程说明 74
2.7.4 示例代码 74
2.8 录像下载转码 81
2.8.1 简介 81
2.8.2 接口总览 81
2.8.3 流程说明 81
2.8.4 示例代码 82
2.9 云台控制 90
2.9.1 简介 90

2.9.2 接口总览 90
2.9.3 流程说明 90
2.9.4 示例代码 91
2.10 语音对讲 103
2.10.1 简介 103
2.10.2 接口总览 103
2.10.3 流程说明 104
2.10.4 示例代码 108
2.11 视频抓图 128
2.11.1 简介 128
2.11.2 接口总览 128
2.11.3 流程说明 129
2.11.4 示例代码 131
2.12 报警上报 138
2.12.1 简介 138
2.12.2 接口总览 138
2.12.3 流程说明 138
2.12.4 示例代码 139
2.13 设备搜索 146
2.13.1 简介 146
2.13.2 接口总览 146
2.13.3 流程说明 146
2.13.4 示例代码 148
2.14 智能事件上报与抓图 157
2.14.1 简介 157
2.14.2 接口总览 157
2.14.3 流程说明 158
2.14.4 示例代码 159
2.15 APP 管理 167
2.15.1 简介 167
2.15.2 接口总览 167
2.15.3 流程说明 167
2.15.4 示例代码 169
第 3 章 回调函数定义 177
3.1 断线回调函数 fDisConnect 177
3.2 断线重连成功回调函数 fHaveReConnect 177
3.3 实时预览数据回调函数 fRealDataCallBackEx 178
3.4 实时预览数据回调函数 fRealDataCallBackEx2 179
3.5 实时预览转码数据回调函数 fDataCallBackEx 180
3.6 回放进度回调函数 fDownLoadPosCallBack 181
3.7 录像回放数据回调函数 fDataCallBack 181
3.8 录像下载进度回调函数 fTimeDownLoadPosCallBack 182
3.9 报警上报回调 fMessCallBack 183
3.10 搜索设备回调函数 fSearchDevicesCB 184
3.11 搜索设备扩展回调函数 fSearchDevicesCBEx 185
3.12 智能图片报警回调函数 fAnalyzerDataCallBack 185
3.13 视频抓图回调函数 fSnapRev 186
3.14 视频预览断开回调函数 fRealPlayDisConnect 187
3.15 音频数据回调函数 pfAudioDataCallBack 188
第 4 章 结构体定义 189
4.1 设备信息结构体 NET_DEVICEINFO 189

4.2 设置登录相关参数结构体 NET_PARAM 189
4.3 设备信息扩展结构体 NET_DEVICEINFO_Ex 191
4.4 登录接口入参结构体 NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY 192
4.5 登录接口出参结构体 NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY 192
4.6 搜索接口入参结构体 NET_IN_STARTSERACH_DEVICE 193
4.7 搜索接口出参结构体 NET_OUT_STARTSERACH_DEVICE 194
4.8 视频数据帧信息结构体 tagVideoFrameParam 194
4.9 音频数据帧信息结构体 tagCBPCMDataParam 195
4.10 时间结构体 NET_TIME 196
4.11 录像文件信息结构体 NET_RECORDFILE_INFO 197
4.12 云台能力集信息结构体 CFG_PTZ_PROTOCOL_CAPS_INFO 199
4.13 云台转动角度范围结构体 CFG_PTZ_MOTION_RANGE 202
4.14 灯光控制内容结构体 CFG_PTZ_LIGHTING_CONTROL 202
4.15 设备支持的语音对讲类型 DHDEV_TALKFORMAT_LIST 203
4.16 语音编码信息结构体 DHDEV_TALKDECODE_INFO 203
4.17 系统信息结构体 DHDEV_SYSTEM_ATTR_CFG 203
4.18 对讲参数结构体 NET_SPEAK_PARAM 205
4.19 语音对讲的转发模式结构体 NET_TALK_TRANSFER_PARAM 206
4.20 设备搜索回调信息结构体 DEVICE_NET_INFO_EX 206
4.21 手动抓拍入参结构体 MANUAL_SNAP_PARAMETER 208
4.22 权限信息扩展结构体 OPR_RIGHT_EX 209
4.23 权限信息结构体 OPR_RIGHT_NEW 209
4.24 设备通道数量信息结构体 NET_DEV_CHN_COUNT_INFO 209
4.25 通道数量信息结构体 NET_CHN_COUNT_INFO 210
4.26 获取抓图配置能力入参结构体 NET_IN_SNAP_CFG_CAPS 210
4.27 获取抓图配置能力出参结构体 NET_OUT_SNAP_CFG_CAPS 211
4.28 图片分辨率结构体 DH_RESOLUTION_INFO 211
4.29 视频编码参数结构体 CFG_VIDEOENC_OPT 212
4.30 视频格式结构体 CFG_VIDEO_FORMAT 213
4.31 音频格式结构体 CFG_AUDIO_ENCODE_FORMAT 216
4.32 多区域遮挡配置结构体 CFG_VIDEO_COVER 217
4.33 遮挡信息结构体 CFG_COVER_INFO 218
4.34 区域信息结构体 CFG_RECT 218
4.35 图像通道属性结构体 CFG_ENCODE_INFO 219
4.36 抓图参数结构体 SNAP_PARAMS 220
4.37 设备软件版本结构体 DH_VERSION_INFO 221
4.38 开始实时预览转码入参 NET_IN_REALPLAY_BY_DATA_TYPE 222
4.39 开始实时预览转码出参 NET_OUT_REALPLAY_BY_DATA_TYPE 223
4.40 开始录像回放转码入参 NET_IN_PLAYBACK_BY_DATA_TYPE 223
4.41 开始录像回放转码出参 NET_OUT_PLAYBACK_BY_DATA_TYPE 224
4.42 开始录像下载转码入参 NET_IN_DOWNLOAD_BY_DATA_TYPE 225
4.43 开始录像下载转码出参 NET_OUT_DOWNLOAD_BY_DATA_TYPE 226
4.44 DSP 能力描述结构体 DH_DSP_ENCODECAP 226
4.45 获取设备升级状态 DHDEV_UPGRADE_STATE_INFO 229
4.46 启动程序入参结构体 NET_IN_START_APP 229
4.47 启动程序出参结构体 NET_OUT_START_APP 229
4.48 APP 查询配置信息结构体 CFG_DEV_DISPOSITION_INFO 230
4.49 获取安装的应用列表入参结构体 NET_IN_GET_INSTALLED_APP_INFO 231
4.50 获取安装的应用列表出参结构体 NET_OUT_GET_INSTALLED_APP_INFO 231
4.51 升级准备入参结构体 NET_IN_INSTALL_PREPAREEX 232
4.52 升级准备出参结构体 NET_OUT_INSTALL_PREPAREEX 233

4.53 推送升级数据入参结构体 NET_IN_INSTALL_APPEND_DATA 233
4.54 推送升级数据出参结构体 NET_OUT_INSTALL_APPEND_DATA 234
4.55 执行升级入参结构体 NET_IN_INSTALL_EXECUTE 234
4.56 执行升级出参结构体 NET_OUT_INSTALL_EXECUTE 234
4.57 停止程序运行入参结构体 NET_IN_STOP_APP 234
4.58 停止程序运行出参结构体 NET_OUT_STOP_APP 235
4.59 卸载程序入参结构体 NET_IN_INSTALL_EXECUTE 235
4.60 卸载程序出参结构体 NET_OUT_INSTALL_EXECUTE 235
第 5 章 枚举定义 237
5.1 设备类型枚举 NET_DEVICE_TYPE 237
5.2 优化获取硬盘信息接口枚举 EM_OPTIMIZE_TYPE 239
5.3 登录方式类型枚举 EM_LOGIN_SPAC_CAP_TYPE 239
5.4 预览类型枚举 DH_RealPlayType 240
5.5 录像查询类型枚举 EM_QUERY_RECORD_TYPE 241
5.6 设备工作模式类型枚举 EM_USEDEV_MODE 242
5.7 焦距模式类型枚举 EM_SUPPORT_FOCUS_MODE 242
5.8 通用云台控制命令枚举 DH_PTZ_ControlType 243
5.9 云台控制扩展命令枚举 DH_EXTPTZ_ControlType 243
5.10 语音编码类型枚举 DH_TALK_CODING_TYPE 245
5.11 设备控制类型枚举 CtrlType 245
5.12 视频压缩格式类型枚举 CFG_VIDEO_COMPRESSION 247
5.13 码流控制模式枚举 CFG_BITRATE_CONTROL 247
5.14 画质类型枚举 CFG_IMAGE_QUALITY 247
5.15 H264 编码级别枚举 CFG_H264_PROFILE_RANK 248
5.16 音频编码模式枚举 CFG_AUDIO_FORMAT 248
5.17 搜索类型枚举 EM_SEND_SEARCH_TYPE 248
5.18 视频预览断开事件类型 EM_REALPLAY_DISCONNECT_EVENT_TYPE 249
5.19 升级安装执行操作 EM_NET_UPGRADE_INSTALL_TYPE 249
5.20 APP 的运行状态 EM_APP_RUNNING_STATE 249
5.21 APP 的证书状态 EM_APP_LICENSE_STATE 250
5.22 APP 的调试状态 EM_APP_DEBUG_STATE 250
5.23 升级安装准备操作 EM_NET_NEXT_OPERATE 250
5.24 升级安装执行操作 EM_NET_UPGRADE_INSTALL_TYPE 251
第 6 章 接口函数定义 252
6.1 SDK 初始化接口 CLIENT_Init 252
6.2 SDK 清理接口 CLIENT_Cleanup 253
6.3 获取 SDK 的版本信息接口 CLIENT_GetSDKVersion 254
6.4 获取错误码接口 CLIENT_GetLastError 254
6.5 设置断线重连成功回调函数接口 CLIENT_SetAutoReconnect 255
6.6 设置连接设备超时时间和尝试次数接口 CLIENT_SetConnectTime 256
6.7 设置登录网络环境接口 CLIENT_SetNetworkParam 257
6.8 优化获取硬盘信息方案接口 CLIENT_SetOptimizeMode 257
6.9 高安全级别登录接口 CLIENT_LoginWithHighLevelSecurity 258
6.10 登出接口 CLIENT_Logout 259
6.11 开始实时预览接口 CLIENT_RealPlayEx 259
6.12 开启实时预览转码接口 CLIENT_RealPlayByDataType 260
6.13 设置实时预览数据回调接口 CLIENT_SetRealDataCallBackEx 261
6.14 停止实时预览接口 CLIENT_StopRealPlayEx 264
6.15 打开录像查询接口 CLIENT_FindFile 264
6.16 查找录像文件接口 CLIENT_FindNextFile 266
6.17 关闭录像查询接口 CLIENT_FindClose 267

6.18 按时间回放接口 CLIENT_PlayBackByTimeEx 267
6.19 开启录像回放转码接口 CLIENT_PlayBackByDataType 269
6.20 停止录像回放接口 CLIENT_StopPlayBack 271
6.21 获取回放 OSD 时间接口 CLIENT_GetPlayBackOsdTime 271
6.22 查询时间段内录像文件接口 CLIENT_QueryRecordFile 272
6.23 按时间下载录像接口 CLIENT_DownloadByTimeEx 274
6.24 开启录像回放转码接口 CLIENT_DownloadByDataType 276
6.25 停止录像下载接口 CLIENT_StopDownload 278
6.26 按文件回放接口 CLIENT_PlayBackByRecordFileEx 279
6.27 暂停录像回放接口 CLIENT_PausePlayBack 281
6.28 定位录像回放起始点接口 CLIENT_SeekPlayBack 281
6.29 录像快放接口 CLIENT_FastPlayBack 282
6.30 录像慢放接口 CLIENT_SlowPlayBack 282
6.31 恢复正常播放接口 CLIENT_NormalPlayBack 283
6.32 按文件下载录像接口 CLIENT_DownloadByRecordFileEx 284
6.33 解析配置信息接口 CLIENT_ParseData 285
6.34 私有云台控制接口 CLIENT_DHPTZControlEx2 286
6.35 新系统能力查询接口 CLIENT_QueryNewSystemInfo 287
6.36 设置设备工作模式接口 CLIENT_SetDeviceMode 289
6.37 异步搜索设备接口 CLIENT_StartSearchDevicesEx 290
6.38 查询设备状态接口 CLIENT_QueryDevState 291
6.39 打开语音对讲接口 CLIENT_StartTalkEx 292
6.40 停止语音对讲接口 CLIENT_StopTalkEx 292
6.41 开始 PC 端录音接口 CLIENT_RecordStartEx 293
6.42 结束 PC 端录音接口 CLIENT_RecordStopEx 293
6.43 发送语音数据接口 CLIENT_TalkSendData 294
6.44 解码音频数据接口 CLIENT_AudioDecEx 295
6.45 设置报警回调函数接口 CLIENT_SetDVRMessCallBack 295
6.46 订阅报警扩展接口 CLIENT_StartListenEx 296
6.47 停止订阅报警 CLIENT_StopListen 297
6.48 停止异步搜索接口 CLIENT_StopSearchDevices 297
6.49 同步搜索设备接口 CLIENT_SearchDevicesByIPs 298
6.50 订阅智能图片报警接口 CLIENT_RealLoadPictureEx 299
6.51 设备控制扩展接口 CLIENT_ControlDeviceEx 301
6.52 取消订阅智能图片报警接口 CLIENT_StopLoadPic 302
6.53 查询录像下载进度接口 CLIENT_GetDownloadPos 303
6.54 设置抓图回调函数接口 CLIENT_SetSnapRevCallBack 303
6.55 抓图请求接口 CLIENT_SnapPictureEx 304
6.56 启动程序 CLIENT_StartApp 305
6.57 查询配置信息 CLIENT_GetNewDevConfig 306
6.58 获取安装的应用列表 CLIENT_GetInstalledAppInfo 306
6.59 升级准备 CLIENT_UpgraderInstall 307
6.60 推送升级数据 CLIENT_UpgraderInstall 308
6.61 执行升级 CLIENT_UpgraderInstall 309
6.62 停止程序运行 CLIENT_StopApp 310
6.63 卸载程序 CLIENT_RemoveApp 311
附录 1 法律声明 312
附录 2 网络安全建议 313

第 1 章 内容简介

1.1 概述

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、智能设备等产品监控联网应用时的开发套件。

本开发套件主要包括以下功能：

设备登录、实时预览、录像回放和回放控制、录像下载、云台控制、语音对讲、视频抓图、报警上报、设备搜索、智能事件上报与抓图、用户管理以及其他功能（设备重启、设备升级、设备校时、视频图像参数设置、通道名称设置和设备网络参数配置）。

根据环境不同，开发包包含的文件会不同，具体如下。

- 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	播放库
dhnetsdk 辅助库	lvsDrawer.dll	图像显示库
	StreamConvertor.dll	转码库

- Linux 开发包所包含的文件，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 环境要求

- 推荐内存：不低于 512M。

- SDK 支持的系统：
  - ◇ Windows
    Windows 10/Windows 8.1/Windows 7 以及 Windows Server 2008/2003。
  - ◇ Linux
    Red Hat/SUSE 等通用 Linux 系统。

表1-3 各功能适用的设备

功能	支持的设备
设备登录	支持 DVR、NVR、IPC 和球机等。
实时预览	支持 DVR、NVR、IPC 和球机等。
录像回放与回放控制	支持 DVR 和 NVR 等存储设备。
录像下载	支持 DVR 和 NVR 等存储设备。
云台控制	支持球机。
语音对讲	支持 DVR、NVR、IPC 和球机等。
视频抓图	支持 DVR、NVR、IPC 和球机等。
报警上报	支持 DVR、NVR、IPC 和球机等。
设备搜索	支持 DVR、NVR、IPC 和球机等。
智能事件上报与抓图	支持 IVS、车载和智能球机等。

第 2 章 主要功能

说明

本文中所有示例代码仅在 Windows7 操作系统下 vs2005sp1 测试过。

2.1 SDK 初始化

2.1.1 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，本节所述接口本身不与设备做任何交互，负责 SDK 的初始化，设置一些影响全局业务的属性。

注意事项

- 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，不支持多线程调用。
- 使用完毕后需要调用 SDK 清理接口以释放资源。

2.1.2 接口总览

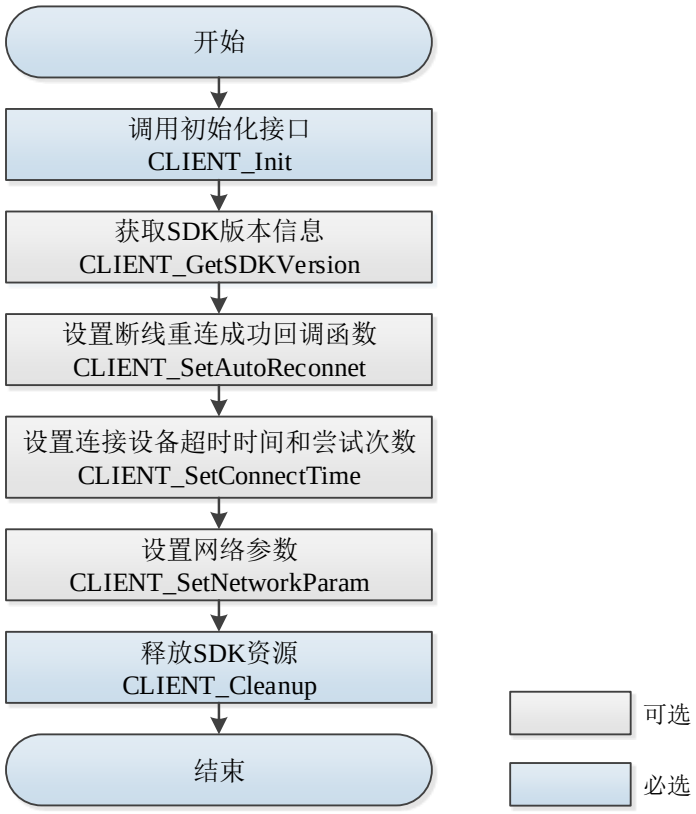
表2-1 SDK 初始化接口说明

接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口,释放 SDK 资源
CLIENT_GetSDKVersion	获取 SDK 版本信息接口
CLIENT_GetLastError	获取接口执行错误码接口
CLIENT_SetAutoReconnect	设置断线重连回调接口
CLIENT_SetConnectTime	设置连接设备超时时间和尝试次数接口
CLIENT_SetNetworkParam	设置登录网络环境接口

2.1.3 流程说明

SDK 初始化业务流程如图 2-1 所示。

图2-1 SDK 初始化业务流程



流程说明

步骤1　完成 SDK 初始化流程。
步骤2　调用 CLIENT_GetSDKVersion，获取 SDK 版本信息（可选）。
步骤3　调用 CLIENT_SetAutoReconnect，设置断线重连回调函数（可选，建议调用），设置后 SDK 内部断线自动重连。
步骤4　调用 CLIENT_SetConnectTime，设置连接设备超时时间和尝试次数（可选）。
步骤5　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（可选）。
步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.1.4 示例代码

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;

//************************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在 SDK 的回调函数中调用 SDK 接口
```

```
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);
//////////////////////////////////////////////////////////////////////////
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

// 此操作为可选操作
// 获取 SDK 版本信息
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作。
    // 此操作为可选操作，但建议用户进行设置。
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;       // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
```

```
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
    CLIENT_SetNetworkParam(&stuNetParm);

    // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，所以建议登录后等待一小段时间，具体等待时间因设备而异。
    Sleep(1000);
    printf("\n");
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
    // 功能业务实现处

}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();

    // 此处可实现退出设备操作

    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}
int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}
```

```
//*************************************************************************************
// 常用回调集合定义
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
	printf("Call DisConnectFunc\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
	printf("Call HaveReConnect\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}
```

2.2 设备登录

2.2.1 简介

设备登录，即用户注册，是进行其他业务的前提。

在 SDK 完成初始化以后，需要用户先登录设备，产生唯一的登录 ID 后才可进行其他业务。登录 ID 是识别本次登录的标志，后续其他功能的 SDK 接口需要传入登录 ID 才可执行。所有业务完成后，需要登出设备，即注销用户。

前提条件

在进行设备登录业务前，您需要先完成 SDK 的初始化。

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，直至成功登录设备或者返回密码错误退出重连功能。

说明

用户可以调用 SDK 自带的断线重连，也可以在应用层调用登录和登出接口手动控制断线重连业务。

注意事项

- SDK 提供的登录业务，无法用于登录其他厂商的设备。请谨慎使用，否则会造成设备无法成功登录。
- 登录和登出配对使用，需要调用登出接口完成用户登出以及 SDK 资源的释放，以免造成资源泄露。
- NVR6 系等设备（支持 16 个及以上硬盘），由于硬盘个数较多，会导致登录业务耗时较长。在进行设备登录业务前，推荐使用 CLIENT_SetOptimizeMode 接口优化获取硬盘信息。登录接口返回的硬盘个数参数，在设置上述优化后将无效，可以通过 CLIENT_QueryDevState（DH_DEVSTATE_DISK）接口获取。优化获取硬盘信息，示例代码如下：

```
int opt = OPTTYPE_MOBILE_DISK_INFO;
CLIENT_SetOptimizeMode(EM_OPT_TYPE_MOBILE_OPTION, &opt);
```

2.2.2 接口总览

表2-2 设备登录接口说明

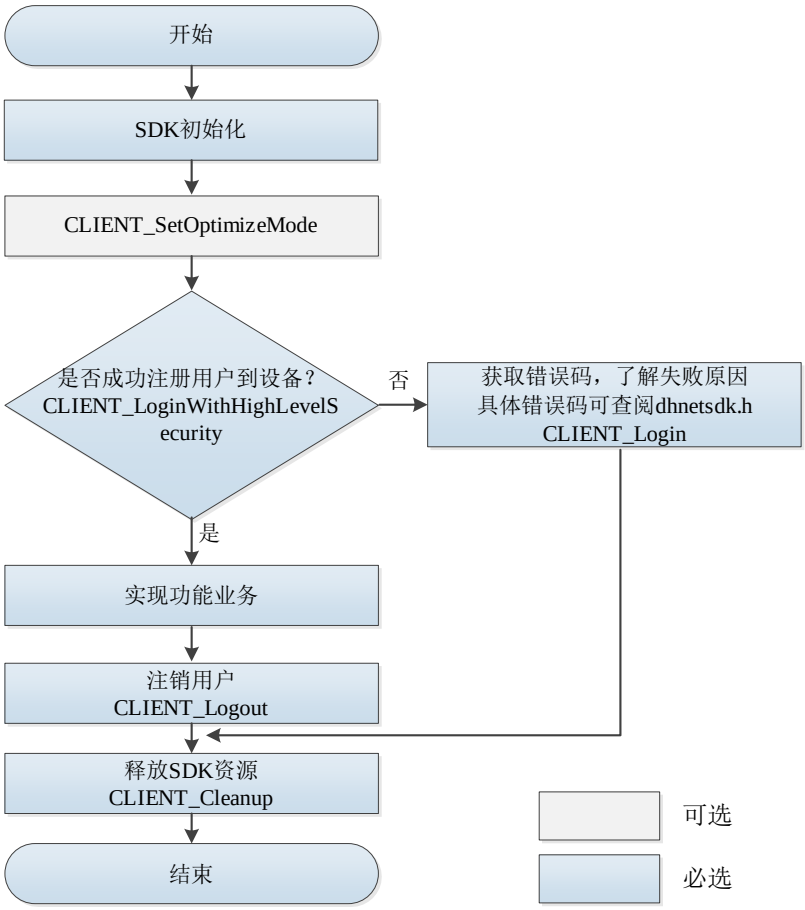
接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_LoginEx2 仍然可以使用，但存在安全风险。所以强烈推荐使用最新接口 CLIENT_LoginWithHighLevelSecurity 登录设备。
CLIENT_Logout	登出接口。
CLIENT_GetLastError	获取接口调用失败时的错误码接口。
CLIENT_SetOptimizeMode	优化获取硬盘信息

2.2.3 流程说明

当 SDK 所在客户端与设备互通时，即可登录设备。当接口返回有效的登录 ID，表示登录设备成功。

登录业务流程如图 2-2 所示。

图2-2 登录业务流程



流程说明

步骤1　完成 SDK 初始化流程。
步骤2　可选操作，优化获取硬盘信息，调用 CLIENT_SetOptimizeMode 接口。
步骤3　初始化成功后，调用 CLIENT_LoginWithHighLevelSecurity 登录设备。
步骤4　登录成功后，用户可以实现需要的业务功能。
步骤5　业务使用完后，调用 CLIENT_Logout 退出设备。
步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.2.4 示例代码

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static LLONG g_lLoginHandle = 0L;
static char g_szDevIp[32] = "192.168.1.25";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
```

```
static BOOL g_bNetSDKInitFlag = FALSE;

//*****************************************************************************
// 常用回调集合声明
// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

//*****************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);


    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);
```

```
        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;          // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与
CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
    CLIENT_SetNetworkParam(&stuNetParm);
    // 此操作为可选操作，优化获取硬盘信息
    int opt = OPTTYPE_MOBILE_DISK_INFO;
    CLIENT_SetOptimizeMode(EM_OPT_TYPE_MOBILE_OPTION, &opt);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if (0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头
文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误
码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" ,
g_szDevIp , g_nPort , CLIENT_GetLastError());
```

```
            }
            else
            {
                printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp ,
g_nPort);
            }
            // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一
小段时间，具体等待时间因设备而异
            Sleep(1000);
            printf("\n");
        }
}
void RunTest()
{
    // 功能业务实现处
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if (FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

int main()
```

```
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}
//**************************************************************************************
// 常用回调集合定义
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```

2.3 实时预览

2.3.1 简介

实时预览，即向存储设备或前端设备获取实时预览数据的功能，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、辅码流和第三码流等。

- 支持配置前端设备的码流分辨率、编码方式以及码率等参数。
- 支持设置图像饱和度、对比度和曝光度等属性。

- 支持用户传入窗口句柄，SDK 直接进行码流解析及播放。
- 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，用户可以自行保存回调码流实现，也可以调用 SDK 接口实现。

2.3.2 接口总览

表2-3 实时预览接口说明

接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_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_RealPlayEx	开始实时预览扩展接口
CLIENT_StopRealPlayEx	停止实时预览扩展接口
CLIENT_SetRealDataCallBackEx	设置实时预览数据回调函数扩展接口
CLIENT_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.3.3 流程说明

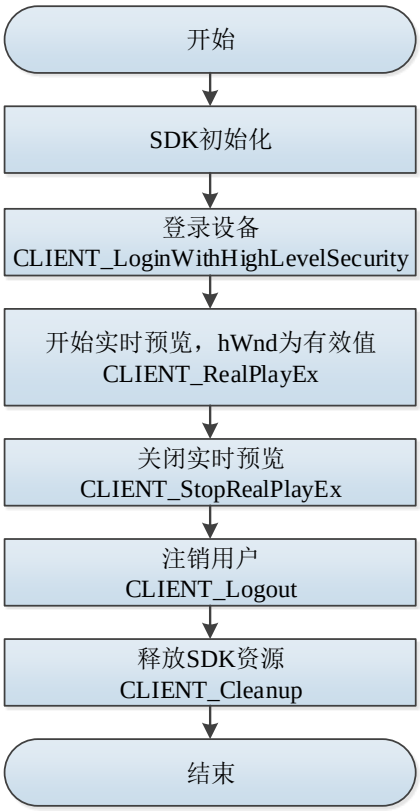
实时监控的业务有以下两种实现方式：

- SDK 解码播放
  SDK 通过调用辅助库里的 playsdk 库实现实时播放。
- 第三方解码播放
  SDK 只回调实时预览码流给用户，用户通过第三方库进行解码播放。

2.3.3.1 SDK 解码播放

SDK 解码播放流程如图 2-3 所示。

图2-3 SDK 解码播放流程



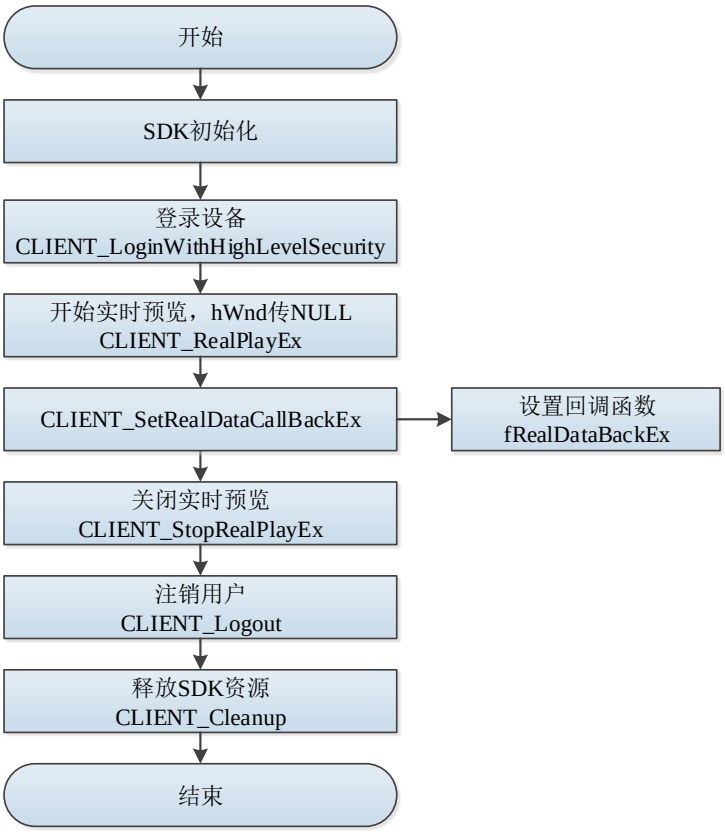
流程说明

步骤1　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　登录成功后，调用 CLIENT_RealPlayEx 启动实时预览，参数 hWnd 为有效窗口句柄。
步骤4　实时预览使用完毕后，调用 CLIENT_StopRealPlayEx 停止实时预览。
步骤5　业务使用完后，调用 CLIENT_Logout 退出设备。
步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.3.3.2 第三方解码播放

第三方解码播放流程如图 2-4 所示。

图2-4 第三方解码播放流程



流程说明

步骤1　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　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_SetRealDataCallBackEx 设置实时数据回调函数。
步骤5　在回调函数中保存实时数据，用于后期处理。强烈不建议在回调中做除了数据传存以外的处理，否则在预览路数较多时会严重影响性能。
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 退出设备。
步骤8　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.3.4 示例代码

2.3.4.1 SDK 解码播放

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

typedef HWND (WINAPI *PROCGETCONSOLEWINDOW)();
PROCGETCONSOLEWINDOW GetConsoleWindow;

static BOOL g_bNetSDKInitFlag = FALSE;
```

```
static LLONG g_lLoginHandle = 0L;
static LLONG g_lRealHandle = 0;
static char g_szDevIp[32] = "192.168.1.10 ";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//*****************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用
该函数
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser);

//*****************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);
```

```
    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;        // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

   NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
```

```
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" ,
g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp ,
g_nPort);
        }
        // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一
小段时间，具体等待时间因设备而异
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    // 判断是否初始化成功
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
    // 判断是否登录设备
    if (0 == g_lLoginHandle)
    {
        return;
    }

    // 实现实时预览功能业务
    // 获取控制台窗口句柄
    HMODULE hKernel32 = GetModuleHandle("kernel32");
    GetConsoleWindow =
(PROCGETCONSOLEWINDOW)GetProcAddress(hKernel32,"GetConsoleWindow");
    HWND hWnd = GetConsoleWindow();

    printf("user can input any key to quit during real play!\n");
    Sleep(1000);

    //开启实时预览
    int nChannelID = 0; // 预览通道号
    DH_RealPlayType emRealPlayType =    DH_RType_Realplay; // 实时预览
    g_lRealHandle = CLIENT_RealPlayEx(g_lLoginHandle, nChannelID, hWnd, emRealPlayType);
    if (0 == g_lRealHandle)
```

```
    {
        printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 关闭预览
    if (0 != g_lRealHandle)
    {
        if(FALSE == CLIENT_StopRealPlayEx(g_lRealHandle))
        {
            printf("CLIENT_StopRealPlayEx Failed!Last Error[%x]\n" , CLIENT_GetLastError());
        }
        else
        {
            g_lRealHandle = 0;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}

int main()
{
```

```
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//****************************************************************************
// 常用回调集合定义
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```

2.3.4.2 第三方解码播放

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")
```

```
static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lRealHandle = 0;
static char g_szDevIp[32] = "192.168.1.10";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//**************************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 实时预览数据回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetRealDataCallBackEx 设置该回调函数，当收到实时预览数据时，SDK 会调用该函数
// 建议用户在此回调函数中只进行保存数据的操作，不建议用户在回调函数里直接对数据进行编解码等处理
// 即：将相应的数据拷贝到自己的存储空间，离开回调函数后再对数据做编解码等处理
void CALLBACK RealDataCallBackEx(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser);

//**************************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
```

```
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);


    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;     // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;          // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
    CLIENT_SetNetworkParam(&stuNetParm);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
```

```
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if (0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一小段时间，具体等待时间因设备而异
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }

    // 实现实时预览功能业务
    printf("user can input any key to quit during real play data callback!\n");
    Sleep(1000);

    //开启实时预览
```

```
    int nChannelID = 0; // 预览通道号
    DH_RealPlayType emRealPlayType =    DH_RType_Realplay; // 实时预览
    g_lRealHandle = CLIENT_RealPlayEx(g_lLoginHandle, nChannelID, NULL, emRealPlayType);
    if (0 == g_lRealHandle)
    {
        printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
        return;
    }
    else
    {
        DWORD dwFlag = 0x00000001;
        if (FALSE == CLIENT_SetRealDataCallBackEx(g_lRealHandle, &RealDataCallBackEx,
NULL, dwFlag))
        {
            printf("CLIENT_SetRealDataCallBackEx: failed! Error code: %x.\n",
CLIENT_GetLastError());
            return;
        }
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 关闭预览
    if (0 != g_lRealHandle)
    {
        if (FALSE == CLIENT_StopRealPlayEx(g_lRealHandle))
        {
            printf("CLIENT_StopRealPlayEx Failed, g_lRealHandle[%x]!Last Error[%x]\n" ,
g_lRealHandle, CLIENT_GetLastError());
        }
        else
        {
            g_lRealHandle = 0;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
```

```
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//**********************************************************************************
// 常用回调集合定义
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
```

```
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK RealDataCallBackEx(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer,
DWORD dwBufSize, LONG param, LDWORD dwUser)
{
    // 若多个实时预览使用相同的数据回调函数，则用户可通过 lRealHandle 进行一一对应
    if (lRealHandle == g_lRealHandle)
    {
        switch(dwDataType)
        {
            case 0:
                //原始音视频混合数据
                printf("receive real data, param: lRealHandle[%p], dwDataType[%d],
pBuffer[%p], dwBufSize[%d], param[%p], dwUser[%p]\n",
                    lRealHandle, dwDataType, pBuffer, dwBufSize, param, dwUser);

                break;
            case 1:
                //标准视频数据

                break;
            case 2:
                //yuv 数据

                break;
            case 3:
                //pcm 音频数据

                break;
            case 4:
                //原始音频数据

                break;
            default:
                break;
```

```
        }

    }
}
```

2.4 录像回放

2.4.1 简介

录像回放是指客户端远程播放设备中指定时间段内的录像文件，寻找所需要的视频信息。

回放功能支持多种回放操作，如正常播放、暂停、快放、慢放、拖动播放等。

根据用户选择不同的解码方式，录像回放方式包括 SDK 解码回放和第三方解码回放两种，关于两种回放方式的详细流程请参见“2.4.3 流程说明”。

2.4.2 接口总览

表2-4 录像回放接口说明

接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_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_PlayBackByTimeEx	按时间方式回放--扩展接口
CLIENT_SetDeviceMode	设置设备工作模式(语音对讲、回放、权限等)
CLIENT_StopPlayBack	停止录像回放接口
CLIENT_GetPlayBackOsdTime	获取回放 OSD（视频浓缩）时间接口
CLIENT_PausePlayBack	暂停或恢复录像回放
CLIENT_FastPlayBack	快放接口，将当前帧率提高一倍
CLIENT_SlowPlayBack	慢放，将当前帧率降低一倍
CLIENT_NormalPlayBack	恢复正常播放速度接口
CLIENT_SeekPlayBack	定位录像回放起始点
CLIENT_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.4.3 流程说明

根据用户选择不同的解码方式，录像回放可以分为以下两种：

- SDK 解码回放
  即用户输入录像的起始时间、结束时间和有效窗口句柄，SDK 调用相应的解码库解析码流，并在视频显示窗口上展示。
- 第三方解码回放
  即用户输入录像的起始时间、结束时间、有效窗口句柄（窗口句柄传 NULL）和有效的回放码流回调函数。SDK 在收到回放的码流数据后，通过码流回调函数将回放码流回调给用户，由用户将码流保存起来，离开回调函数后用户再调用第三方库对已保存的码流数据进行解析和显示。

2.4.3.1 SDK 解码回放

SDK 解码回放流程，如图 2-5 所示。

图2-5 SDK 解码回放流程



流程说明

步骤1　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　登录成功后，两次调用 CLIENT_SetDeviceMode，分别设置回放时的码流类型和回放时的录像文件类型。

步骤4　调用 CLIENT_PlayBackByTimeEx 启动回放，hWnd 参数为有效窗口句柄值。

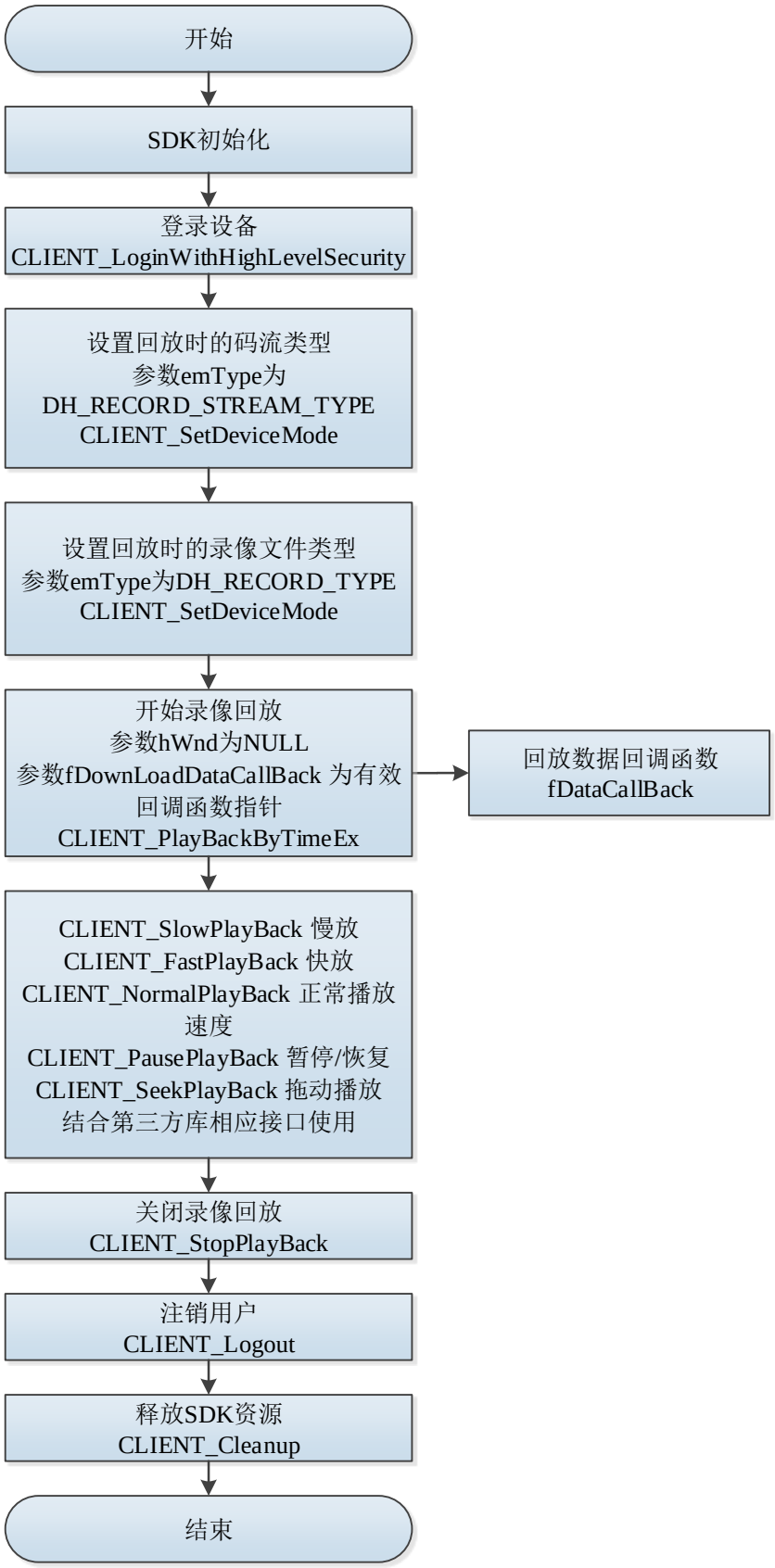
步骤5　回放过程中，根据用户需求调用 CLIENT_SlowPlayBack 实现慢放，CLIENT_FastPlayBack 实现快放，CLIENT_NormalPlayBack 实现正常播放速度，CLIENT_PausePlayBack 实现暂停/恢复，CLIENT_SeekPlayBack 实现拖动播放。

步骤6　回放使用完毕后，调用 CLIENT_StopPlayBack 停止回放。
步骤7　业务使用完后，调用 CLIENT_Logout 注销用户。
步骤8　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.4.3.2 第三方解码回放

第三方库解码回放流程如图 2-6 所示。

图2-6 第三方解码回放流程



流程说明

步骤1　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　登录成功后，两次调用 CLIENT_SetDeviceMode，分别设置回放时的码流类型和回放时

的录像文件类型。

步骤4 登录成功后，调用 CLIENT_PlayBackByTimeEx 启动回放，hWnd 参数为 NULL，fDownLoadDataCallBack 参数为有效回调函数指针。

步骤5 SDK 收到回放数据后，通过设置的 fDownLoadDataCallBack 回调函数回调回放数据给用户，用户在回调函数中保存回放数据，离开回调函数后，再通过第三方库解码回放录像数据。

步骤6 回放过程中，根据用户需求调用 CLIENT_SlowPlayBack 实现慢放，CLIENT_FasePlayBack 实现快放, CLIENT_NormalPlayBack 实现正常播放速度，CLIENT_PausePlayBack 实现暂停/恢复，CLIENT_SeekPlayBack 实现拖动播放，同时需调用第三方库相应的接口。

步骤7 回放使用完毕后，调用 CLIENT_StopPlayBack 停止回放。

步骤8 业务使用完后，调用 CLIENT_Logout 注销用户。

步骤9 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.4.4 示例代码

2.4.4.1 SDK 解码回放

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"
#pragma comment(lib , "dhnetsdk.lib")

extern "C" HWND WINAPI GetConsoleWindow();

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lPlayHandle = 0L;
static char g_szDevIp[32] = "192.168.1.13";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//************************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数。
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数。
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
```

```
dwUser);

//*************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }
    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;        // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
```

```
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23)                // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // 用户初次登录设备，可能要初始化一些数据才能正常实现业务功能，所以建议登录后等待一小段时间，具体等待时间因设备而异
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
```

```
        return;
    }

    // 录像回放功能
    // 获取控制台窗口句柄
    HWND hWnd = GetConsoleWindow();

    // 设置回放时的码流类型
    int nStreamType = 0; // 0-主辅码流,1-主码流,2-辅码流
    CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

    // 设置回放时的录像文件类型
    NET_RECORD_TYPE emFileType = NET_RECORD_TYPE_ALL; // 所有录像
    CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_TYPE, &emFileType);

    //开启录像回放
    int nChannelID = 0; // 通道号
    NET_TIME stuStartTime = {0};
    stuStartTime.dwYear = 2015;
    stuStartTime.dwMonth = 11;
    stuStartTime.dwDay = 20;

    NET_TIME stuStopTime = {0};
    stuStopTime.dwYear = 2015;
    stuStopTime.dwMonth = 11;
    stuStopTime.dwDay = 21;

    g_lPlayHandle = CLIENT_PlayBackByTimeEx(g_lLoginHandle, nChannelID, &stuStartTime,
&stuStopTime, hWnd, NULL, NULL, NULL, NULL);
    if (0 == g_lPlayHandle)
    {
        printf("CLIENT_PlayBackByTimeEx: failed! Error code: %x.\n", CLIENT_GetLastError());
    }

    // 用户可根据需求实现回放控制
    // 由于这个是控制台 demo，无法将录像回放和回放控制同时展现给用户，这里提供示例代
码用于参考

    // CLIENT_SlowPlayBack 实现慢放
    /* 示例代码
    if (FALSE == CLIENT_SlowPlayBack (g_lPlayHandle))
    {
        printf("CLIENT_SlowPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" ,
```

```
g_lPlayHandle, CLIENT_GetLastError());
    }
    */

    // CLIENT_FastPlayBack 实现快放
    /* 示例代码
    if (FALSE == CLIENT_FastPlayBack (g_lPlayHandle))
    {
        printf("CLIENT_FastPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
    }
    */

    // CLIENT_NormalPlayBack 实现正常播放速度
    /* 示例代码
    if (FALSE == CLIENT_NormalPlayBack (g_lPlayHandle))
    {
        printf("CLIENT_NormalPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
    }
    */

    // CLIENT_PausePlayBack 实现暂停/恢复
    /* 示例代码
    if (FALSE == CLIENT_PausePlayBack (g_lPlayHandle, TRUE))
    {
        printf("CLIENT_PausePlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
    }
    */

    // CLIENT_SeekPlayBack 实现拖动播放
    /* 示例代码
    int nOffsetSeconds = 2 * 60 * 60; // 拖动至 stuStartTime 后 2*60*60 秒的位置开始回放
    if (FALSE == CLIENT_SeekPlayBack (g_lPlayHandle, nOffsetSeconds, 0))
    {
        printf("CLIENT_SeekPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
    }
    */
}

void EndTest()
```

```
{
    printf("input any key to quit!\n");
    getchar();
    // 关闭回放
    if (0 != g_lPlayHandle)
    {
        if (FALSE == CLIENT_StopPlayBack(g_lPlayHandle))
        {
            printf("CLIENT_StopPlayBack Failed, g_lRealHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
        }
        else
        {
            g_lPlayHandle = 0;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
```

```
}

//**************************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
	printf("Call DisConnectFunc\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
	printf("Call HaveReConnect\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}
```

2.4.4.2 第三方解码回放

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"
#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lPlayHandle = 0L;
static char g_szDevIp[32] = "192.168.1.10";
```

```
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//**************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数。
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数。
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 回放进度回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_PlayBackByTimeEx 设置该回调函数，当接收到设备端的回放数据时，SDK 会调用该函数。
void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);

// 回放数据回调函数
// 不建议在该回调函数中调用 SDK 接口
// 当设置该回调函数时，若对应的 hWnd 参数为 NULL，参数返回，0：表示本次回调失败，下次回调会返回相同的数据，1：表示本次回调成功，下次回调会返回后续的数据
// 当设置该回调函数时，若对应的 hWnd 参数不为 NULL，则不管回调函数返回值为多少都认为回调成功，下次回调会返回后续的数据
// 通过 CLIENT_PlayBackByTimeEx 设置该回调函数，当接收到设备端的回放数据时，SDK 会调用该函数。
int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);

//**************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
```

```
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }
    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;       // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);
```

```
    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23)                // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // 用户初次登录设备，可能要初始化一些数据才能正常实现业务功能，所以建议登录后等待一小段时间，具体等待时间因设备而异
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }

    // 录像回放功能

    // 设置回放时的码流类型
```

```
int nStreamType = 0; // 0-主辅码流,1-主码流,2-辅码流
CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

// 设置回放时的录像文件类型
NET_RECORD_TYPE emFileType = NET_RECORD_TYPE_ALL; // 所有录像
CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_TYPE, &emFileType);

//开启录像回放
int nChannelID = 0; // 通道号
NET_TIME stuStartTime = {0};
stuStartTime.dwYear = 2015;
stuStartTime.dwMonth = 11;
stuStartTime.dwDay = 20;

NET_TIME stuStopTime = {0};
stuStopTime.dwYear = 2015;
stuStopTime.dwMonth = 11;
stuStopTime.dwDay = 21;

// 函数形参 hWnd 需为 NULL
// 函数形参 fDownLoadDataCallBack 需为 有效回调函数指针
g_lPlayHandle = CLIENT_PlayBackByTimeEx(g_lLoginHandle, nChannelID, &stuStartTime,
&stuStopTime, NULL, &DownLoadPosCallBack, NULL, &DataCallBack, NULL);
if (g_lPlayHandle == 0)
{
    printf("CLIENT_PlayBackByTimeEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}

// 用户可根据需求实现回放控制
// 由于是第三方库解码，用户在调用 SDK 的回放控制接口的同时，还需要调用第三方库相
应的控制接口
// 由于这个是控制台 demo，无法将录像回放和回放控制同时展现给用户，这里提供示例代
码用于参考

// CLIENT_SlowPlayBack 实现慢放
/* 示例代码
if (FALSE == CLIENT_SlowPlayBack (g_lPlayHandle))
{
    printf("CLIENT_SlowPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
}
// 第三方库相应接口调用
```

```
    */

    // CLIENT_FastPlayBack 实现快放
    /* 示例代码
    if (FALSE == CLIENT_FastPlayBack (g_lPlayHandle))
    {
        printf("CLIENT_FastPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
    }
    // 第三方库相应接口调用

    */

    // CLIENT_NormalPlayBack 实现正常播放速度
    /* 示例代码
    if (FALSE == CLIENT_NormalPlayBack (g_lPlayHandle))
    {
        printf("CLIENT_NormalPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
    }
    // 第三方库相应接口调用

    */

    // CLIENT_PausePlayBack 实现暂停/恢复
    /* 示例代码
    if (FALSE == CLIENT_PausePlayBack (g_lPlayHandle, TRUE))
    {
        printf("CLIENT_PausePlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
    }
    // 第三方库相应接口调用

    */

    // CLIENT_SeekPlayBack 实现拖动播放
    /* 示例代码
    int nOffsetSeconds = 2 * 60 * 60; // 拖动至 stuStartTime 后 2*60*60 秒的位置开始回放
    if (FALSE == CLIENT_SeekPlayBack (g_lPlayHandle, nOffsetSeconds, 0))
    {
        printf("CLIENT_SeekPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
    }
```

```
    // 第三方库相应接口调用

    */
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 关闭回放
    if (0 != g_lPlayHandle)
    {
        if (FALSE == CLIENT_StopPlayBack(g_lPlayHandle))
        {
            printf("CLIENT_StopPlayBack Failed, g_lRealHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
        }
        else
        {
            g_lPlayHandle = 0;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}
```

```
int main()
{
	InitTest();

	RunTest();

	EndTest();

	return 0;
}

//****************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
	printf("Call DisConnectFunc\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
	printf("Call HaveReConnect\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}

void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
```

```
dwDownLoadSize, LDWORD dwUser)
{
    // 若多个回放/下载使用相同的进度回调函数，则用户可通过 lPlayHandle 进行一一对应
    if (lPlayHandle == g_lPlayHandle)
    {
        printf("lPlayHandle[%p]\n", lPlayHandle);
        printf("dwTotalSize[%d]\n", dwTotalSize);
        printf("dwDownLoadSize[%d]\n", dwDownLoadSize);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
    }
}

int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
dwBufSize, LDWORD dwUser)
{
    int nRet = 0;
    printf("call DataCallBack\n");
    // 若多个回放/下载使用相同的数据回调函数，则用户可通过 lRealHandle 进行一一对应
    if(lRealHandle == g_lPlayHandle)
    {
        BOOL bSuccess = TRUE;
        // 以下打印在回放/下载时会造成刷屏现象，请注意
        printf("lPlayHandle[%p]\n", lRealHandle);
        printf("dwDataType[%d]\n", dwDataType);
        printf("pBuffer[%p]\n", pBuffer);
        printf("dwBufSize[%d]\n", dwBufSize);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
        switch(dwDataType)
        {
        case 0:
            //Original data
            // 用户在此处保存码流数据，离开回调函数后再进行解码或转发等一系列处理
            nRet = 1;//

            break;
        case 1:
            //Standard video data

            break;
        case 2:
            //yuv data
```

```
            break;
        case 3:
            //pcm audio data

            break;
        case 4:
            //Original audio data

            break;
        default:
            break;
        }
    }
    return nRet;
}
```

2.5 录像下载

2.5.1 简介

录像下载，即用户通过 SDK 获取存储设备上存有的录像并保存到本地的过程。允许用户对当前所选通道的录像进行下载，并可将视频导出到本地硬盘或者外接设备 U 盘等。

录像下载方式

录像下载方式包括按文件下载和按时间下载两种，关于两种下载方式的详细流程请参见“2.5.3 流程说明”。

2.5.2 接口总览

表2-5 录像下载接口说明

接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_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_SetDeviceMode	设置设备工作模式(语音对讲、回放、权限等)
CLIENT_QueryRecordFile	查询时间段内的所有录像文件的接口
CLIENT_FindFile	打开录像查询句柄接口
CLIENT_FindNextFile	查找录像文件接口
CLIENT_FindClose	关闭录像查询句柄接口
CLIENT_DownloadByRecordFileEx	按文件下载录像扩展接口
CLIENT_DownloadByTimeEx	按时间下载录像扩展接口
CLIENT_GetDownloadPos	查询录像下载进度

接口	接口说明
CLIENT_StopDownload	停止录像下载接口
CLIENT_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.5.3 流程说明

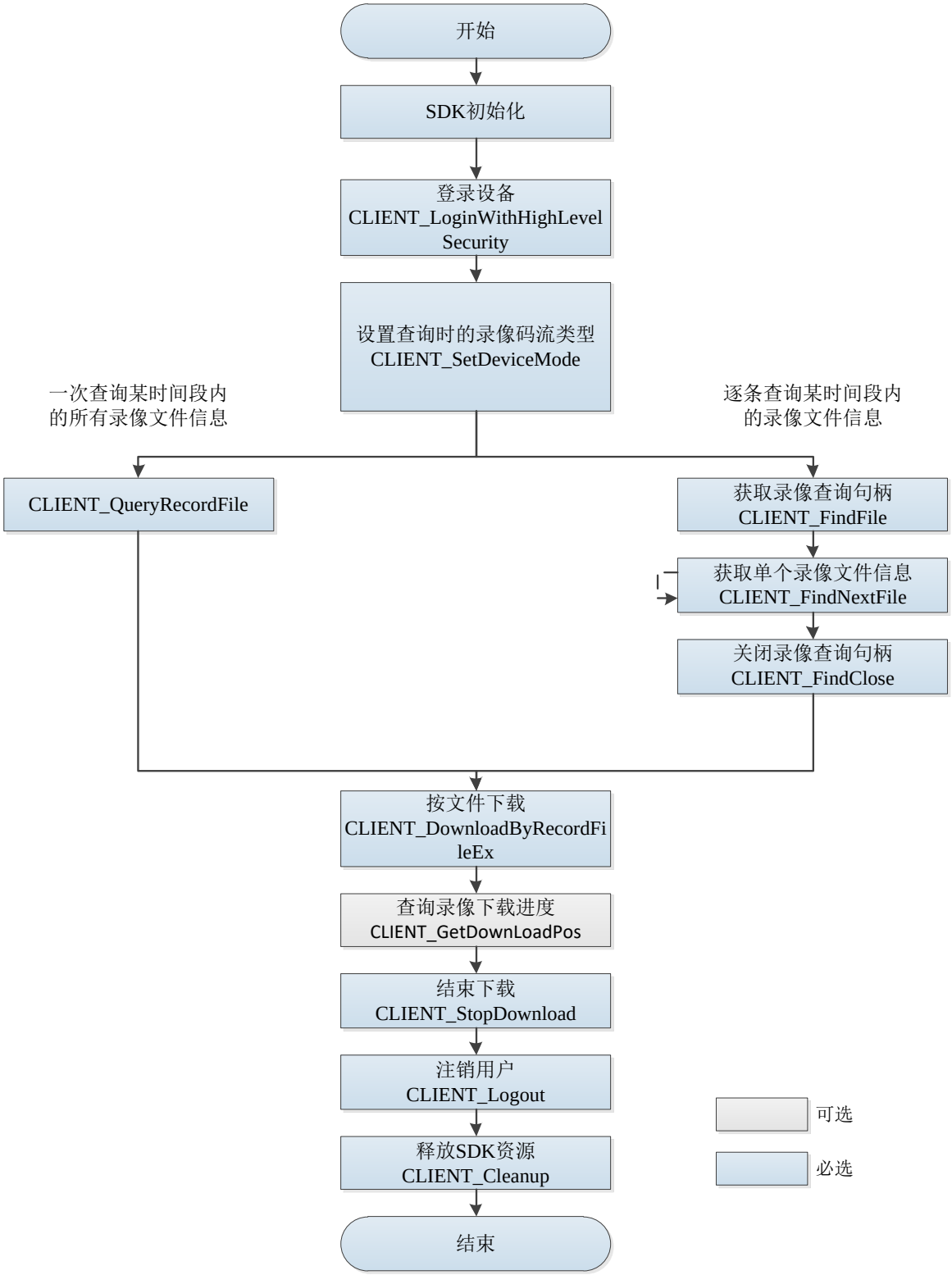
录像下载方式包括以下两种：

- 按文件下载
  即用户传入需要下载的录像文件信息，SDK 可将指定的录像文件下载并保存到用户指定的文件中。同时，用户也可以提供一个回调函数的指针，SDK 将指定的录像文件的数据通过回调函数回调给用户，由用户自行处理。
- 按时间下载
  即用户传入需要下载的起始时间和结束时间，SDK 可将指定时间段内的录像文件下载并保存到用户指定的文件中。同时，用户也可以提供一个回调函数的指针，SDK 将指定时间段内的录像数据通过回调函数回调给用户，由用户自行处理。

2.5.3.1 按文件下载

按文件下载录像流程如图 2-7 所示。

图2-7 按文件下载录像流程



流程说明

步骤1　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_SetDeviceMode 设置录像查询时的录像码流类型，对应 emType 为 DH_RECORD_STREAM_TYPE，建议设置为“0-主辅码流”，否则在少数设备上会无法获得结果。如果只需要主码流录像，可以在结果中滤除辅码流录像信息。具体请参见“第 5 章 枚举定义”中的 EM_USEDEV_MODE 枚举说明。

步骤4　可通过以下两种方式查询录像文件：

- 调用 CLIENT_FindFile 获取录像查询句柄，再循环调用 CLIENT_FindNextFile 接口，逐次获取下一个录像文件信息，最后调用 CLIENT_FindClose 关闭录像查询句柄。
- 调用 CLIENT_QueryRecordFile 一次性获取某时间段内的所有录像文件信息。

步骤5　获取到录像文件信息后，调用 CLIENT_DownloadByRecordFileEx 开始录像文件下载，形参 sSavedFileName 和 fDownLoadDataCallBack 中至少有一个需为有效值。

步骤6　下载过程中，根据用户需求调用 CLIENT_GetDownloadPos 查询录像下载进度。

步骤7　录像下载完毕后，调用 CLIENT_StopDownload 停止下载。

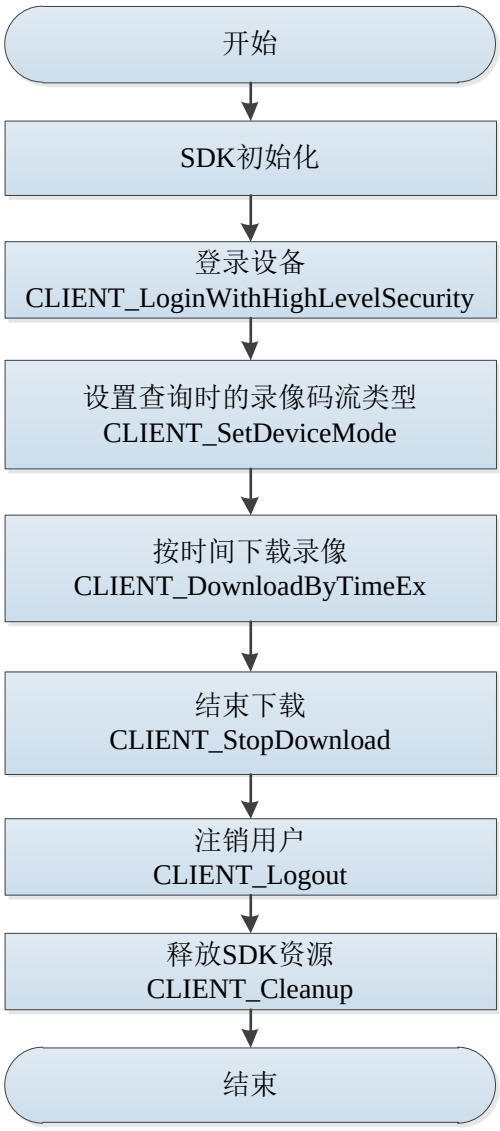
步骤8　业务使用完后，调用 CLIENT_Logout 退出设备。

步骤9　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.5.3.2 按时间下载

按时间下载流程如图 2-8 所示。

图2-8 按时间下载流程



流程说明

步骤1　调用 SDK 初始化模块。

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

步骤3　调用 CLIENT_SetDeviceMode 设置录像查询时的录像码流类型，对应 emType 为 DH_RECORD_STREAM_TYPE，具体请参见“第 5 章枚举定义”中的 EM_USEDEV_MODE

枚举说明。

步骤4 调用 CLIENT_DownloadByTimeEx 接口开始按时间下载，形参 sSavedFileName 和 fDownLoadDataCallBack 中至少有一个需为有效值。

步骤5 录像下载完毕后，调用 CLIENT_StopDownload 停止下载。根据需要可以等文件下载完了关闭下载，也可以下载到一部分停止下载。

步骤6 业务使用完后，调用 CLIENT_Logout 退出设备。

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

2.5.4 示例代码

2.5.4.1 按文件下载

```
#include <windows.h>
#include <stdio.h>
#include <vector>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lDownloadHandle = 0L;
static char g_szDevIp[32] = "192.168.1.10";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
static const int g_nMaxRecordFileCount = 5000;
//*****************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 回放/下载 进度回调函数
// 不建议在该回调函数中调用 SDK 接口
```

```
// dwDownLoadSize: -1 时表示本次回放/下载结束，-2 表示写文件失败，其他值表示有效数据
// 通过 CLIENT_DownloadByRecordFileEx 设置该回调函数，当 SDK 收到回放/下载数据时，SDK 会调用该函数
void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);

// 回放/下载 数据回调函数
// 不建议在该回调函数中调用 SDK 接口
// 回放时：参数返回，0：表示本次回调失败，下次回调会返回相同的数据，1：表示本次回调成功，下次回调会返回后续的数据
// 下载时：不管回调函数返回值为多少都认为回调成功，下次回调会返回后续的数据
// 通过 CLIENT_DownloadByRecordFileEx 设置该回调函数，当 SDK 收到回放/下载数据时，SDK 会调用该函数
int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);

//*************************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);


    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);


        // 设置登录超时时间和尝试次数
```

```
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;          // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与
CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
    CLIENT_SetNetworkParam(&stuNetParm);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if (0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头
文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误
码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" ,
g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp ,
```

```
g_nPort);
            }
            // 用户初次登录设备，可能要初始化一些数据才能正常实现业务功能，所以建议登录后等待一小段时间，具体等待时间因设备而异。
            Sleep(1000);
            printf("\n");
        }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }
    // 录像文件查询
    // 设置查询时的录像码流类型
    int nStreamType = 0; // 0-主辅码流,1-主码流,2-辅码流
    CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

    // 录像查询有两种实现方式：1，一次取完时间段内的所有录像文件；2，分次取时间段内的所有录像文件。
    // 此处通过第二种方案实现，第一种方案的实现可参考 CLIENT_QueryRecordFile 接口说明。
    int nChannelID = 0; // 通道号
    NET_TIME stuStartTime = {0};
    stuStartTime.dwYear = 2015;
    stuStartTime.dwMonth = 9;
    stuStartTime.dwDay = 20;

    NET_TIME stuStopTime = {0};
    stuStopTime.dwYear = 2015;
    stuStopTime.dwMonth = 9;
    stuStopTime.dwDay = 30;
    int lFindHandle = CLIENT_FindFile(g_lLoginHandle, nChannelID, 0, NULL, &stuStartTime, &stuStopTime, FALSE, 5000);
    if (0 == lFindHandle)
    {
        printf("CLIENT_FindFile Failed!Last Error[%x]\n",CLIENT_GetLastError());
```

```
        return;
    }
    // demo 的示例代码，以最大支持 g_nMaxRecordFileCount 录像文件为例。
    std::vector<NET_RECORDFILE_INFO> bufFileInfo(g_nMaxRecordFileCount);

    for (int nFileIndex = 0; nFileIndex < g_nMaxRecordFileCount; ++nFileIndex)
    {
        int result = CLIENT_FindNextFile(lFindHandle, &bufFileInfo[nFileIndex]);
        if (0 == result)// 录像文件信息数据取完
        {
            break;
        }
        else if (1 != result)// 参数出错
        {
            printf("CLIENT_FindNextFile Failed!Last Error[%x]\n",CLIENT_GetLastError());
            break;
        }
    }

    //停止查找
    if(0 != lFindHandle)
    {
        CLIENT_FindClose(lFindHandle);
    }
    // 将查询过来的第一个文件设置为下载文件
    NET_RECORDFILE_INFO stuNetFileInfo;
    if (nFileIndex > 0)
    {
        memcpy(&stuNetFileInfo, (void *)&bufFileInfo[0], sizeof(stuNetFileInfo));
    }
    else
    {
        printf("no record, return\n");
        return;
    }

    // 录像文件下载

    // 开启录像下载
    // 函数形参 sSavedFileName 和 fDownLoadDataCallBack 至少有一个为有效值
    // 实际应用中，一般根据需求选择直接保存至 sSavedFileName 或回调处理数据两者之一
    g_lDownloadHandle = CLIENT_DownloadByRecordFileEx(g_lLoginHandle, &stuNetFileInfo,
"test.dav", DownLoadPosCallBack, NULL, DataCallBack, NULL);
```

```
    if (0 == g_lDownloadHandle)
    {
        printf("CLIENT_DownloadByRecordFileEx: failed! Error code: %x.\n",
CLIENT_GetLastError());
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 关闭下载，可在下载结束后调用，也可在下载中调用。
    if (0 != g_lDownloadHandle)
    {
        if (FALSE == CLIENT_StopDownload(g_lDownloadHandle))
        {
            printf("CLIENT_StopDownload Failed, g_lDownloadHandle[%x]!Last Error[%x]\n" ,
g_lDownloadHandle, CLIENT_GetLastError());
        }
        else
        {
            g_lDownloadHandle = 0;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
```

```
    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//****************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
```

```
dwDownLoadSize, LDWORD dwUser)
{
    // 若多个回放/下载使用相同的进度回调函数，则用户可通过 lPlayHandle 进行一一对应
    if (lPlayHandle == g_lDownloadHandle)
    {
        printf("lPlayHandle[%p]\n", lPlayHandle);
        printf("dwTotalSize[%d]\n", dwTotalSize);
        printf("dwDownLoadSize[%d]\n", dwDownLoadSize);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
    }
}

int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
dwBufSize, LDWORD dwUser)
{
    int nRet = 0;
    printf("call DataCallBack\n");
    // 若多个回放/下载使用相同的数据回调函数，则用户可通过 lRealHandle 进行一一对应
    if(lRealHandle == g_lDownloadHandle)
    {
        printf("lPlayHandle[%p]\n", lRealHandle);
        printf("dwDataType[%d]\n", dwDataType);
        printf("pBuffer[%p]\n", pBuffer);
        printf("dwBufSize[%d]\n", dwBufSize);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
        switch(dwDataType)
        {
        case 0:
            //Original data
            // 用户在此处保存码流数据，离开回调函数后再进行解码或转发等一系列处理
            nRet = 1;

            break;
        case 1:
            //Standard video data

            break;
        case 2:
            //yuv data

            break;
```

```
            case 3:
                //pcm audio data

                break;
            case 4:
                //Original audio data

                break;
            default:
                break;
            }
        }
        return nRet;
}
```

2.5.4.2 按时间下载

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lDownloadHandle = 0L;
static char g_szDevIp[32] = "192.168.1.10";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//************************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数。
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用
该函数。
```

```
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 按时间回放进度回调函数
// 不建议在该回调函数中调用 SDK 接口
// dwDownLoadSize: -1 时表示本次回放/下载结束，-2 表示写文件失败，其他值表示有效数据
// 通过 CLIENT_DownloadByTimeEx 设置该回调函数，当 SDK 收到回放/下载数据时，SDK 会调用该函数
void CALLBACK TimeDownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser);

// 回放/下载 数据回调函数
// 不建议在该回调函数中调用 SDK 接口
// 回放时：参数返回，0：表示本次回调失败，下次回调会返回相同的数据，1：表示本次回调成功，下次回调会返回后续的数据
// 下载时：不管回调函数返回值为多少都认为回调成功，下次回调会返回后续的数据
// 通过 CLIENT_DownloadByTimeEx 设置该回调函数，当 SDK 收到回放/下载数据时，SDK 会调用该函数
int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);

//*************************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);


    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
```

```
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;        // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与
CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头
文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误
码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" ,
g_szDevIp , g_nPort , CLIENT_GetLastError());
```

```
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp ,
g_nPort);
        }
        // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一
小段时间，具体等待时间因设备而异。
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }
    // 录像文件查询
    // 设置查询时的录像码流类型
    int nStreamType = 0; // 0-主辅码流,1-主码流,2-辅码流
    CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

    int nChannelID = 0; // 通道号
    NET_TIME stuStartTime = {0};
    stuStartTime.dwYear = 2015;
    stuStartTime.dwMonth = 9;
    stuStartTime.dwDay = 17;

    NET_TIME stuStopTime = {0};
    stuStopTime.dwYear = 2015;
    stuStopTime.dwMonth = 9;
    stuStopTime.dwDay = 18;

    // 录像下载功能业务实现处

    // 开启录像下载
```

```
    // 函数形参 sSavedFileName 和 fDownLoadDataCallBack 需至少有一个为有效值，否则入参有误
    g_lDownloadHandle = CLIENT_DownloadByTimeEx(g_lLoginHandle, nChannelID, EM_RECORD_TYPE_ALL, &stuStartTime, &stuStopTime, "test.dav", TimeDownLoadPosCallBack, NULL, DataCallBack, NULL);
    if (g_lDownloadHandle == 0)
    {
        printf("CLIENT_DownloadByTimeEx: failed! Error code: %x.\n", CLIENT_GetLastError());
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 关闭下载，可在下载结束后调用，也可在下载中调用
    if (0 != g_lDownloadHandle)
    {
        if (FALSE == CLIENT_StopDownload(g_lDownloadHandle))
        {
            printf("CLIENT_StopDownload Failed, g_lDownloadHandle[%x]!Last Error[%x]\n" , g_lDownloadHandle, CLIENT_GetLastError());
        }
        else
        {
            g_lDownloadHandle = 0;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
```

```
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//**********************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
```

```
}

void CALLBACK TimeDownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser)
{
    // 若多个回放/下载使用相同的进度回调函数，则用户可通过 lPlayHandle 进行一一对应
    if (lPlayHandle == g_lDownloadHandle)
    {
        printf("lPlayHandle[%p]\n", lPlayHandle);
        printf("dwTotalSize[%d]\n", dwTotalSize);
        printf("dwDownLoadSize[%d]\n", dwDownLoadSize);
        printf("index[%d]\n", index);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
    }
}

int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
dwBufSize, LDWORD dwUser)
{
    int nRet = 0;
    printf("call DataCallBack\n");
    // 若多个回放/下载使用相同的数据回调函数，则用户可通过 lRealHandle 进行一一对应
    if(lRealHandle == g_lDownloadHandle)
    {
        printf("lPlayHandle[%p]\n", lRealHandle);
        printf("dwDataType[%d]\n", dwDataType);
        printf("pBuffer[%p]\n", pBuffer);
        printf("dwBufSize[%d]\n", dwBufSize);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
        switch(dwDataType)
        {
        case 0:
            //Original data
            // 用户在此处保存码流数据，离开回调函数后再进行解码或转发等一系列处理
            nRet = 1;//

            break;
        case 1:
            //Standard video data

            break;
```

```
            case 2:
                //yuv data

                break;
            case 3:
                //pcm audio data

                break;
            case 4:
                //Original audio data

                break;
            default:
                break;
            }
        }
        return nRet;
}
```

2.6 实时预览转码

2.6.1 简介

实时预览转码，即向存储设备或前端设备获取实时预览数据，转换成用户需要的码流类型。

- 支持国标 PS 码流。
- 支持 TS 码流。
- 支持 MP4 码流。
- 支持 H264/H265 裸码流。
- 支持 PS 码流。
- 支持 RTP 码流

2.6.2 接口总览

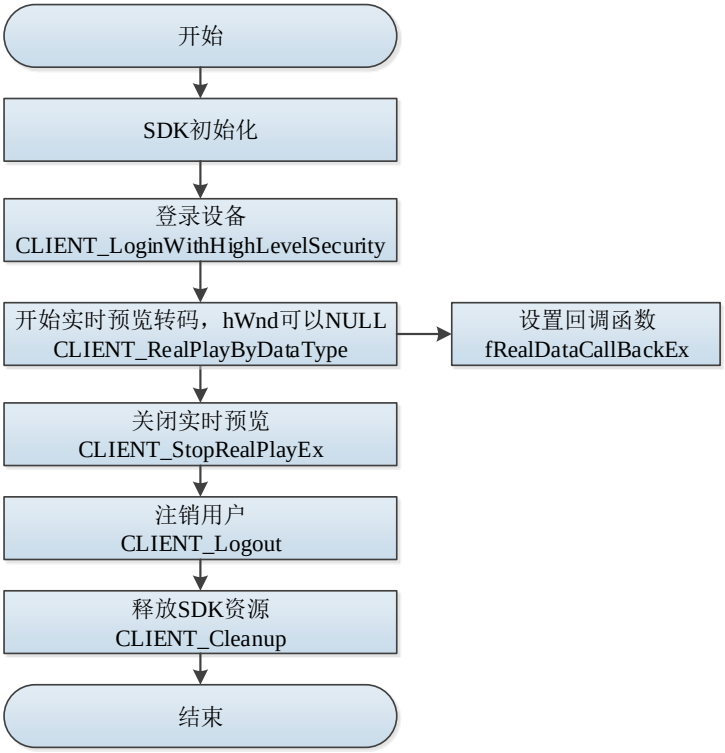
表2-6 实时预览转码接口说明

接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_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_RealPlayByDataType	开始实时预览转码接口
CLIENT_StopRealPlayEx	停止实时预览扩展接口
CLIENT_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.6.3 流程说明

实时预览转码流程如图 2-120 所示。

图2-9 实时预览转码流程



流程说明

步骤1　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　登录成功后，调用 CLIENT_RealPlayByDataType，启动实时预览，参数 hWnd 可以为 NULL。
步骤4　设置实时数据回调函数 fRealDataCallBackEx，保存转码后的数据。
步骤5　实时预览转码使用完毕后，调用 CLIENT_StopRealPlayEx 停止实时预览。
步骤6　业务使用完后，调用 CLIENT_Logout 退出设备。
步骤7　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.6.4 示例代码

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

extern "C" HWND WINAPI GetConsoleWindow();

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lRealHandle = 0;
```

```
static char g_szDevIp[32] = "192.168.1.10 ";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//**************************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用
该函数
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser);

// 通过 CLIENT_RealPlayByDataType 设置该回调函数，当接收到设备端的回放数据时，SDK 会
调用该函数。
int CALLBACK OnDataCallBackEx(LLONG lRealHandle, NET_DATA_CALL_BACK_INFO
*pDataCallBack, LDWORD dwUser);
//**************************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
```

```
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;          // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

   NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
            // 例如：
```

```
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一小段时间，具体等待时间因设备而异
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    // 判断是否初始化成功
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
    // 判断是否登录设备
    if (0 == g_lLoginHandle)
    {
        return;
    }

    //开启实时预览转码
    NET_IN_REALPLAY_BY_DATA_TYPE stIn = {sizeof(stIn)};
    NET_OUT_REALPLAY_BY_DATA_TYPE stOut = {sizeof(stOut)};
    stIn.emDataType = EM_REAL_DATA_TYPE_H264;
    stIn.rType = DH_RType_Realplay;
    stIn.nChannelID = 0;
    stIn.hWnd = NULL;
    stIn.dwUser = NULL;
    stIn.cbRealDataEx2 = OnDataCallBackEx;

    g_lRealHandle = CLIENT_RealPlayByDataType(g_lLoginHandle, &stIn, &stOut, 5000);
    if (0 == g_lRealHandle)
    {
```

```
            printf("CLIENT_RealPlayByDataType: failed! Error code: %x.\n", CLIENT_GetLastError());
        }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 关闭预览
    if (0 != g_lRealHandle)
    {
        if(FALSE == CLIENT_StopRealPlayEx(g_lRealHandle))
        {
            printf("CLIENT_StopRealPlayEx Failed!Last Error[%x]\n" , CLIENT_GetLastError());
        }
        else
        {
            g_lRealHandle = 0;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}

int main()
{
    InitTest();
```

```
    RunTest();
    EndTest();
    return 0;
}

//**************************************************************************************
// 常用回调集合定义
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

int CALLBACK OnDataCallBackEx(LLONG lRealHandle, NET_DATA_CALL_BACK_INFO
*pDataCallBack, LDWORD dwUser)
{
    if(pDataCallBack->dwDataType == (NET_DATA_CALL_BACK_VALUE +
EM_REAL_DATA_TYPE_H264))
    {
        printf("Frame Type subType=%d\n",pDataCallBack->emFramSubType);
        if ( pDataCallBack->emFramSubType != 4 && pDataCallBack->emFramSubType != 11
&& pDataCallBack->emFramSubType != 15)
```

```
        {
            SYSTEMTIME tm;
            GetLocalTime(&tm);
            static SYSTEMTIME    oldTime = tm;

            char buffer[4096] = {0};
            sprintf( buffer, "Frame Type:%02d
LocalTime:%04d-%02d-%02d %02d:%02d:%02d %03d
DevTime:%04d-%02d-%02d %02d:%02d:%02d %d\n",    pDataCallBack->emFramSubType,

    tm.wYear,tm.wMonth,tm.wDay,tm.wHour,tm.wMinute,tm.wSecond,tm.wMilliseconds,
                    pDataCallBack->stuTime.dwYear,
pDataCallBack->stuTime.dwMonth,pDataCallBack->stuTime.dwDay,pDataCallBack->stuTime.d
wHour,pDataCallBack->stuTime.dwMinute,pDataCallBack->stuTime.dwSecond,
pDataCallBack->stuTime.dwMillisecond);

            oldTime = tm;
        }
    }
    return 0;
}
```

2.7 录像回放转码

2.7.1 简介

录像回放是指客户端远程播放设备中指定时间段内的录像文件，寻找所需要的视频信息，转换成用户需要的码流类型。

- 支持国标 PS 码流。
- 支持 TS 码流。
- 支持 MP4 码流。
- 支持 H264/H265 裸码流。
- 支持 PS 码流。
- 支持 RTP 码流

2.7.2 接口总览

表2-7 实时预览转码接口说明

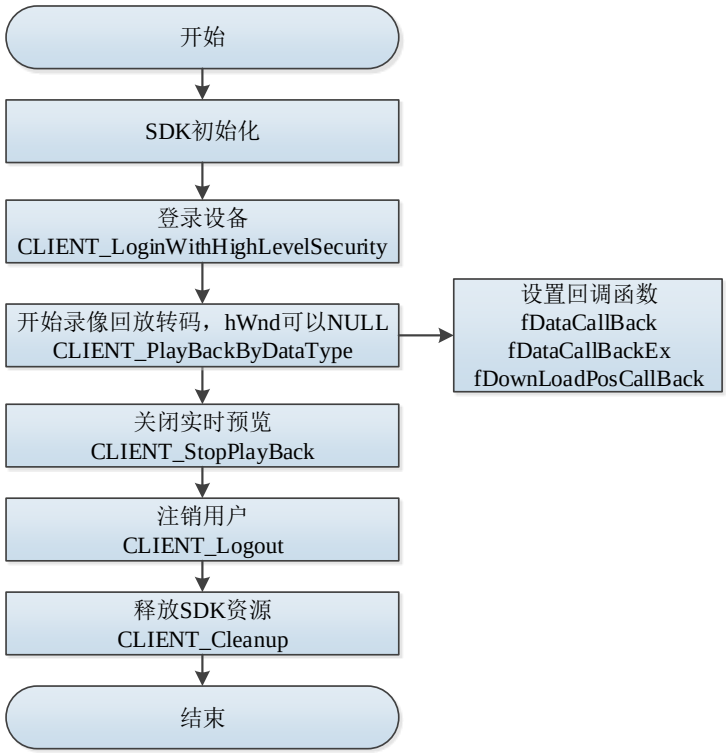
接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_LoginEx2 仍然可以使用，但存在安全风险。所以强烈推荐使用最新接口 CLIENT_LoginWithHighLevelSecurity 登录设备。

接口	接口说明
CLIENT_PlayBackByDataType	开始录像回放转码接口
CLIENT_StopPlayBack	停止录像回放接口
CLIENT_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.7.3 流程说明

实时预览转码流程如图 2-121 所示。

图2-10 录像回放转码流程



流程说明

步骤8　完成 SDK 初始化流程。
步骤9　初始化成功后，调用 CLIENT_LoginWithHighLevelSecurity 登录设备。
步骤10　登录成功后，调用 CLIENT_PlayBackByDataType，启动录像下载，参数 hWnd 可以为 NULL。
步骤11　设置录像回放数据回调函数 fDataCallBackEx，录像回放数据回调函数 fDataCallBack，录像回放进度回调函数 fDownLoadPosCallBack，保存转码后的数据。
步骤12　录像回放转码使用完毕后，调用 CLIENT_StopPlayBack 停止录像回放。
步骤13　业务使用完后，调用 CLIENT_Logout 退出设备。
步骤14　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.7.4 示例代码

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"
#pragma comment(lib , "dhnetsdk.lib")
```

```
extern "C" HWND WINAPI GetConsoleWindow();

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lPlayHandle = 0L;
static char g_szDevIp[32] = "192.168.1.13";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//*************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数。
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数。
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 回放进度回调函数原形
void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);

// 回放数据回调函数原形
int CALLBACK PlayBackDataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);

// 回放数据回调函数原形(扩展)
int CALLBACK DataCallBackEx(LLONG lRealHandle, NET_DATA_CALL_BACK_INFO *pDataCallBack, LDWORD dwUser);
//*************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
```

```
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }
    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;     // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;        // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);
```

```
    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23)                // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // 用户初次登录设备，可能要初始化一些数据才能正常实现业务功能，所以建议登录后等待一小段时间，具体等待时间因设备而异
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }

    NET_IN_PLAYBACK_BY_DATA_TYPE stuIn = {sizeof(stuIn)};
    stuIn.nChannelID = 0;
    stuIn.emDataType = EM_REAL_DATA_TYPE_MP4;
    stuIn.emAudioType = EM_AUDIO_DATA_TYPE_G711A;
    stuIn.hWnd = 0;
```

```
    stuIn.cbDownLoadPos = DownLoadPosCallBack;
    stuIn.fDownLoadDataCallBack = PlayBackDataCallBack;
    stuIn.fDownLoadDataCallBackEx = DataCallBackEx;
    stuIn.dwDataUser = NULL;
    stuIn.stStartTime.dwYear = 2023;
    stuIn.stStartTime.dwMonth = 2;
    stuIn.stStartTime.dwDay = 1;
    stuIn.stStartTime.dwHour = 11;
    stuIn.stStartTime.dwMinute = 10;
    stuIn.stStartTime.dwSecond = 12;

    stuIn.stStopTime.dwYear = 2023;
    stuIn.stStopTime.dwMonth = 2;
    stuIn.stStopTime.dwDay = 1;
    stuIn.stStopTime.dwHour = 11;
    stuIn.stStopTime.dwMinute = 11;
    stuIn.stStopTime.dwSecond = 12;

    NET_OUT_PLAYBACK_BY_DATA_TYPE stuOut = {sizeof(stuOut)};

    g_lPlayHandle = CLIENT_PlayBackByDataType(g_lLoginHandle, &stuIn, &stuOut, 5000);
    if (0 == g_lPlayHandle)
    {
        printf("CLIENT_PlayBackByDataType fail,error:%X\n", CLIENT_GetLastError());
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 关闭回放
    if (0 != g_lPlayHandle)
    {
        if (FALSE == CLIENT_StopPlayBack(g_lPlayHandle))
        {
            printf("CLIENT_StopPlayBack Failed, g_lRealHandle[%x]!Last Error[%x]\n" ,
g_lPlayHandle, CLIENT_GetLastError());
        }
        else
        {
            g_lPlayHandle = 0;
        }
```

```
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//************************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
```

```
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, LDWORD dwUser)
{
    printf("DownLoad Progress:%d/%d\n", dwDownLoadSize, dwTotalSize);
}

int CALLBACK PlayBackDataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer,
DWORD dwBufSize, LDWORD dwUser)
{
    printf("dwDataType:%d, dwBufSize:%d\n", dwDataType, dwBufSize);

    return 0;
}

int CALLBACK DataCallBackEx(LLONG lRealHandle, NET_DATA_CALL_BACK_INFO
*pDataCallBack, LDWORD dwUser)
{
    NET_DATA_CALL_BACK_INFO    stuDataCallBack = *pDataCallBack;

    printf("DataCallBackEx:dwDataType:%d, dwBufSize:%d, emFramType:%d\n",
stuDataCallBack.dwDataType, stuDataCallBack.dwBufSize, stuDataCallBack.emFramType);

    return 0;
}
```

2.8 录像下载转码

2.8.1 简介

录像下载，即用户通过 SDK 获取存储设备上存有的录像并保存到本地的过程。允许用户对当前所选通道的录像按所需的码流类型进行下载，并可将视频导出到本地硬盘或者外接设备 U 盘等。

- 支持国标 PS 码流。
- 支持 TS 码流。
- 支持 MP4 码流。
- 支持 H264/H265 裸码流。
- 支持 PS 码流。
- 支持 RTP 码流

2.8.2 接口总览

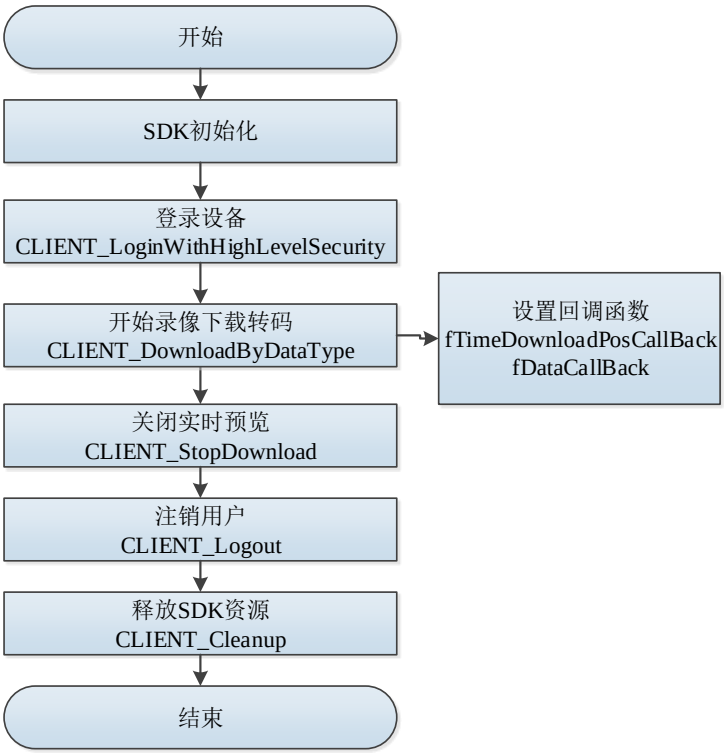
表2-8 实时预览转码接口说明

接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_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_DownloadByDataType	开始录像下载转码接口
CLIENT_StopDownload	停止录像下载接口
CLIENT_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.8.3 流程说明

实时预览转码流程如图 2-122 所示。

图2-11 录像回放转码流程



流程说明

步骤15 完成 SDK 初始化流程。
步骤16 初始化成功后，调用 CLIENT_LoginWithHighLevelSecurity 登录设备。
步骤17 登录成功后，调用 CLIENT_DownloadByDataType，启动录像转码下载，参数 hWnd 可以为 NULL。
步骤18 设置录像下载进度回调函数 fTimeDownloadPosCallBack，录像下载数据回调函数 fDataCallBackEx，保存转码后的数据。
步骤19 录像下载转码使用完毕后，调用 CLIENT_StopDownload 停止录像下载。
步骤20 业务使用完后，调用 CLIENT_Logout 退出设备。
步骤21 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.8.4 示例代码

```
#include <windows.h>
#include <stdio.h>
#include <vector>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lDownloadHandle = 0L;
static char g_szDevIp[32] = "192.168.1.10";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
```

```
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
static const int g_nMaxRecordFileCount = 5000;
char g_szFileName[260] = {0};
//*****************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 回放/下载 进度回调函数
// 不建议在该回调函数中调用 SDK 接口
// dwDownLoadSize: -1 时表示本次回放/下载结束，-2 表示写文件失败，其他值表示有效数据
// 通过 CLIENT_DownloadByDataType 设置该回调函数，当 SDK 收到下载数据时，SDK 会调用该函数
void CALLBACK TimeDownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser);

// 回放/下载 数据回调函数
// 不建议在该回调函数中调用 SDK 接口
// 回放时：参数返回，0：表示本次回调失败，下次回调会返回相同的数据，1：表示本次回调成功，下次回调会返回后续的数据
// 下载时：不管回调函数返回值为多少都认为回调成功，下次回调会返回后续的数据
// 通过 CLIENT_DownloadByDataType 设置该回调函数，当 SDK 收到下载数据时，SDK 会调用该函数
int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);

//*****************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
```

```
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);



    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);



        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;     // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;          // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
    CLIENT_SetNetworkParam(&stuNetParm);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
```

```
    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if (0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // 用户初次登录设备，可能要初始化一些数据才能正常实现业务功能，所以建议登录后等待一小段时间，具体等待时间因设备而异。
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }
    // 录像文件查询
```

```
    // 设置查询时的录像码流类型
    int nStreamType = 0; // 0-主辅码流,1-主码流,2-辅码流
    CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

    // 开启录像下载
    // 函数形参 sSavedFileName 和 fDownLoadDataCallBack 至少有一个为有效值
    // 实际应用中，一般根据需求选择直接保存至 sSavedFileName 或回调处理数据两者之一
    NET_IN_DOWNLOAD_BY_DATA_TYPE stuIn = {sizeof(stuIn)};
    stuIn.nChannelID = 0;
    stuIn.emDataType = EM_REAL_DATA_TYPE_MP4;
    stuIn.emAudioType = EM_AUDIO_DATA_TYPE_G711A;
    stuIn.cbDownLoadPos = TimeDownLoadPosCallBack;
    stuIn.fDownLoadDataCallBack = DataCallBack;
    stuIn.dwDataUser = NULL;
    stuIn.stStartTime.dwYear = 2023;
    stuIn.stStartTime.dwMonth = 2;
    stuIn.stStartTime.dwDay = 1;
    stuIn.stStartTime.dwHour = 11;
    stuIn.stStartTime.dwMinute = 10;
    stuIn.stStartTime.dwSecond = 12;

    stuIn.stStopTime.dwYear = 2023;
    stuIn.stStopTime.dwMonth = 2;
    stuIn.stStopTime.dwDay = 1;
    stuIn.stStopTime.dwHour = 11;
    stuIn.stStopTime.dwMinute = 11;
    stuIn.stStopTime.dwSecond = 12;

    strncpy(g_szFileName, "D:\\file.mp4", sizeof(g_szFileName) - 1);
    stuIn.szSavedFileName = g_szFileName;

    NET_OUT_DOWNLOAD_BY_DATA_TYPE stuOut = {sizeof(stuOut)};
    g_lDownloadHandle = CLIENT_DownloadByDataType(g_lLoginHandle, &stuIn, &stuOut,
5000);
    if (0 == g_lDownloadHandle)
    {
        printf("CLIENT_DownloadByDataType fail,error:%X\n", CLIENT_GetLastError());
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
```

```
    getchar();
    // 关闭下载，可在下载结束后调用，也可在下载中调用。
    if (0 != g_lDownloadHandle)
    {
        if (FALSE == CLIENT_StopDownload(g_lDownloadHandle))
        {
            printf("CLIENT_StopDownload Failed, g_lDownloadHandle[%x]!Last Error[%x]\n" ,
g_lDownloadHandle, CLIENT_GetLastError());
        }
        else
        {
            g_lDownloadHandle = 0;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }

    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}
```

```
//*************************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
	printf("Call DisConnectFunc\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
	printf("Call HaveReConnect\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}

void CALLBACK TimeDownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser)
{
	// 若多个回放/下载使用相同的进度回调函数，则用户可通过 lPlayHandle 进行一一对应
	if (lPlayHandle == g_lDownloadHandle)
	{
		printf("TimeDownLoad Progress:%d/%d\n", dwDownLoadSize, dwTotalSize);
	}
}

int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
```

```
dwBufSize, LDWORD dwUser)
{
    int nRet = 0;
    printf("call DataCallBack\n");
    // 若多个回放/下载使用相同的数据回调函数，则用户可通过 lRealHandle 进行一一对应
    if(lRealHandle == g_lDownloadHandle)
    {
        printf("lPlayHandle[%p]\n", lRealHandle);
        printf("dwDataType[%d]\n", dwDataType);
        printf("pBuffer[%p]\n", pBuffer);
        printf("dwBufSize[%d]\n", dwBufSize);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
        switch(dwDataType)
        {
        case 0:
            //Original data
            // 用户在此处保存码流数据，离开回调函数后再进行解码或转发等一系列处理
            nRet = 1;

            break;
        case 1:
            //Standard video data

            break;
        case 2:
            //yuv data

            break;
        case 3:
            //pcm audio data

            break;
        case 4:
            //Original audio data

            break;
        default:
            break;
        }
    }
    return nRet;
}
```

2.9 云台控制

2.9.1 简介

云台控制是监控系统的重要组成部分之一。在不同的使用场景下，用户有不同的预览需求，比如，在普通的使用场景下，用户需要对监控的画面进行跟踪处理等。用户可通过 SDK 实现对云台设备的一系列控制操作，如上下左右移动、聚焦画面、画面放大缩小、点间巡航和三维定位等。

2.9.2 接口总览

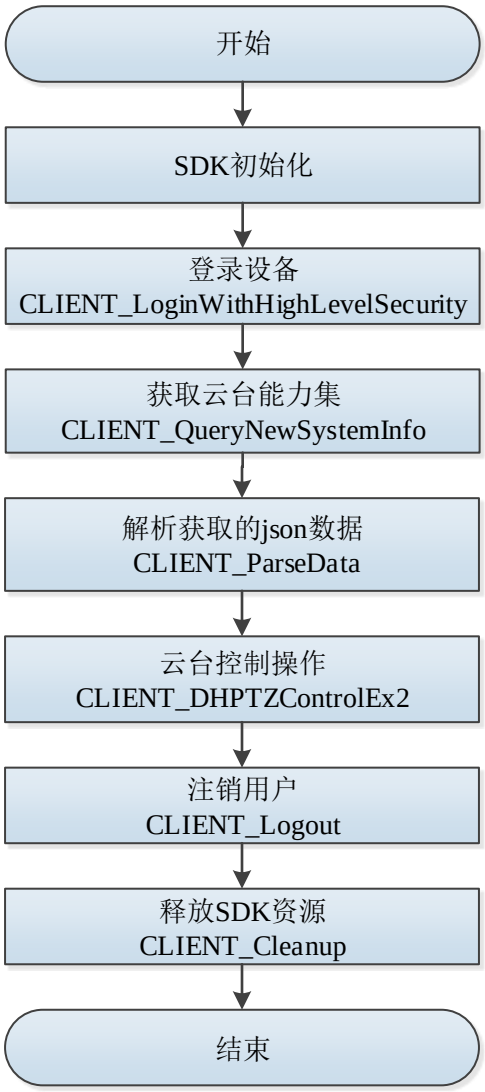
表2-9 云台控制接口说明

接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_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_ParseData	解析查询到的配置信息
CLIENT_DHPTZControlEx2	私有云台控制扩展接口
CLIENT_QueryNewSystemInfo	(新)系统能力查询接口
CLIENT_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.9.3 流程说明

云台控制流程如图 2-12 所示。

图2-12 云台控制流程



流程说明

步骤1　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　登录成功后，调用 CLIENT_QueryNewSystemInfo，命令为 CFG_CAP_CMD_PTZ 获取云台能力集，再调用 CLIENT_ParseData，命令为 CFG_CAP_CMD_PTZ 解析获取的能力集。

步骤4　根据需求调用 CLIENT_DHPTZControlEx2 接口操作云台，不同的云台命令可能需要不同的参数，部分操作命令需要调用相应的停止命令，比如左右移动操作，具体参考示例代码。

步骤5　业务使用完后，调用 CLIENT_Logout 退出设备。

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

2.9.4 示例代码

```
#include <windows.h>
#include <stdio.h>
#include <vector>
#include <string>
#include "dhnetsdk.h"
#include "dhconfigsdk.h"
```

```
#pragma comment(lib , "dhnetsdk.lib")
#pragma comment(lib , "dhconfigsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lDownloadHandle = 0L;
static char g_szDevIp[32] = "171.2.7.34";
static int   g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//**************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数。
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用
该函数。
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser);

//**************************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }
    // 获取 SDK 版本信息
```

```
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;        // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" ,
```

```
g_szDevIp , g_nPort , CLIENT_GetLastError());
            }
            else
            {
                printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp ,
g_nPort);
            }
            // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一
小段时间，具体等待时间因设备而异。
            Sleep(1000);
            printf("\n");
        }
}

// Ptz 控制信息结构体
typedef struct tagPtzControlInfo
{
    tagPtzControlInfo():m_iCmd(-1), m_bStopFlag(false){}
    tagPtzControlInfo(int iCmd, const std::string& sDescription, bool bStopFlag):m_iCmd(iCmd),
m_sDescription(sDescription), m_bStopFlag(bStopFlag){}
    int m_iCmd;
    std::string m_sDescription;
    bool m_bStopFlag; // 部分 Ptz 操作，start 后需要调用相应的 stop 操作
}PtzControlInfo;

// 获取输入的整形
int GetIntInput(char *szPromt, int& nError);

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }

    // 获取云台能力集
    char szBuffer[2048] = "";
```

```
    int nError = 0;
    if (FALSE == CLIENT_QueryNewSystemInfo(g_lLoginHandle, CFG_CAP_CMD_PTZ, 0,
szBuffer, (DWORD)sizeof(szBuffer), &nError))
    {
        printf("CLIENT_QueryNewSystemInfo Failed, cmd[CFG_CAP_CMD_PTZ], Last
Error[%x]\n" , CLIENT_GetLastError());
        return;
    }

    CFG_PTZ_PROTOCOL_CAPS_INFO stuPtzCapsInfo =
{sizeof(CFG_PTZ_PROTOCOL_CAPS_INFO)};
    if (FALSE == CLIENT_ParseData(CFG_CAP_CMD_PTZ, szBuffer, &stuPtzCapsInfo,
sizeof(stuPtzCapsInfo), NULL))
    {
        printf("CLIENT_ParseData Failed, cmd[CFG_CAP_CMD_PTZ], Last Error[%x]\n" ,
CLIENT_GetLastError());
        return;
    }
    // 云台操作
    std::vector<PtzControlInfo> vecPtzControl;
    if (TRUE == stuPtzCapsInfo.bTile)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_UP_CONTROL), "上", true));
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_DOWN_CONTROL), "下", true));
    }

    if (TRUE == stuPtzCapsInfo.bPan)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_LEFT_CONTROL), "左", true));
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_RIGHT_CONTROL), "右", true));
    }

    if (TRUE == stuPtzCapsInfo.bZoom)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_ZOOM_ADD_CONTROL), "变倍+",
true));
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_ZOOM_DEC_CONTROL), "变倍-",
true));
    }

    if (TRUE == stuPtzCapsInfo.bFocus)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_FOCUS_ADD_CONTROL), "调焦+",
true));
```

```
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_FOCUS_DEC_CONTROL), "调焦-",
true));
    }

    if (TRUE == stuPtzCapsInfo.bIris)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_APERTURE_ADD_CONTROL), "光
圈+", true));
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_APERTURE_DEC_CONTROL), "光
圈-", true));
    }

    if (TRUE == stuPtzCapsInfo.bPreset)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_POINT_MOVE_CONTROL), "转至
预置点", false));
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_POINT_SET_CONTROL), "设置预
置点", false));
    }

    if (TRUE == stuPtzCapsInfo.bRemovePreset)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_POINT_DEL_CONTROL), "删除预
置点", false));
    }

    if (TRUE == stuPtzCapsInfo.bTour)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_POINT_LOOP_CONTROL), "点间
巡航", false));
        vecPtzControl.push_back(PtzControlInfo(int(DH_EXTPTZ_ADDTOLOOP), "加入预置点到
巡航", false));
        vecPtzControl.push_back(PtzControlInfo(int(DH_EXTPTZ_DELFROMLOOP), "删除巡航
中预置点", false));
    }

    if (TRUE == stuPtzCapsInfo.bRemoveTour)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_EXTPTZ_CLOSELOOP), "清除巡航",
false));
    }

    if (TRUE == stuPtzCapsInfo.bTile && TRUE == stuPtzCapsInfo.bPan)
    {
```

```
        vecPtzControl.push_back(PtzControlInfo(int(DH_EXTPTZ_LEFTTOP), "左上", true));
        vecPtzControl.push_back(PtzControlInfo(int(DH_EXTPTZ_RIGHTTOP), "右上", true));
        vecPtzControl.push_back(PtzControlInfo(int(DH_EXTPTZ_LEFTDOWN), "左下", true));
        vecPtzControl.push_back(PtzControlInfo(int(DH_EXTPTZ_RIGHTDOWN), "右下", true));
    }

    if (TRUE == stuPtzCapsInfo.bMoveRelatively)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_EXTPTZ_FASTGOTO), "快速定位",
false));
    }

    if (TRUE == stuPtzCapsInfo.bMoveAbsolutely)
    {
        vecPtzControl.push_back(PtzControlInfo(int(DH_EXTPTZ_EXACTGOTO), "三维精确定位",
false));
    }

    vecPtzControl.push_back(PtzControlInfo(int(-2), "暂停", false));
    vecPtzControl.push_back(PtzControlInfo(int(-1), "退出", true));
    PtzControlInfo cLastChoose;
    while(TRUE)
    {
        printf("云台控制操作：\n");
        for (std::vector<PtzControlInfo>::const_iterator iter = vecPtzControl.begin(); iter !=
vecPtzControl.end(); ++iter)
        {
            printf("\t%d\t:%s\n", iter->m_iCmd, iter->m_sDescription.c_str());
        }
        int nError = 0;
        int nChoose = GetIntInput("\t 选择： ", nError);
        if (0 != nError)
        {
            printf("无效输入!\n");
            continue;
        }

        std::vector<PtzControlInfo>::iterator iterFind = vecPtzControl.begin();
        for (; iterFind != vecPtzControl.end(); ++iterFind)
        {
            if (nChoose == iterFind->m_iCmd)
            {
                break;
```

```
            }
        }

        if (iterFind == vecPtzControl.end())
        {
            printf("请输入有效范围内的操作\n");
            continue;
        }
        // 停止上一个操作
        int nChannelId = 0;
        if (true == cLastChoose.m_bStopFlag)
        {
            if (FALSE == CLIENT_DHPTZControlEx2(g_lLoginHandle, nChannelId,
cLastChoose.m_iCmd, 0, 0, 0, TRUE))
            {
                printf("CLIENT_DHPTZControlEx2 Failed, cLastChoose->GetCmd()[%x]!Last
Error[%x]\n" , cLastChoose.m_iCmd, CLIENT_GetLastError());
            }
        }

        if (iterFind->m_sDescription == "暂停")
        {
            cLastChoose = *iterFind;
            continue;
        }

        if (iterFind->m_sDescription == "退出")
        {
            break;
        }
        // 不同的 PTZ 命令对应不同的额外参数设置方案，参数设置指导如下
        // 额外参数
        LONG lParam1 = 0;
        LONG lParam2 = 0;
        LONG lParam3 = 0;
        void* pParam4 = NULL;
        if (DH_PTZ_UP_CONTROL <= iterFind->m_iCmd && iterFind->m_iCmd <=
DH_PTZ_RIGHT_CONTROL)
        {
            // 垂直/水平移动速度，有效范围(1-8)
            lParam2 = 3;
        }
        else if (DH_PTZ_ZOOM_ADD_CONTROL <= iterFind->m_iCmd && iterFind->m_iCmd
```

```
<= DH_PTZ_APERTURE_DEC_CONTROL)
        {
            // 速度，有效范围(1-8)
            lParam1 = 3;
        }
        else if (DH_PTZ_POINT_MOVE_CONTROL <= iterFind->m_iCmd && iterFind->m_iCmd
<= DH_PTZ_POINT_DEL_CONTROL)
        {
            // lParam2 表示预置点号
            printf("\t 预置点号(%2d-%2d):",
stuPtzCapsInfo.wPresetMin,stuPtzCapsInfo.wPresetMax);
            scanf("%d", &lParam2);
        }
        else if (DH_PTZ_POINT_LOOP_CONTROL == iterFind->m_iCmd)
        {
            // lParam1 表示巡航路线，lParam3：76 开始；96 停止
            printf("\t 巡航路线(%2d-%2d):",
stuPtzCapsInfo.wTourMin,stuPtzCapsInfo.wTourMax);
            scanf("%d", &lParam1);
            printf("\t1:开始\n\t2:停止\n\t 选择:");
            int nTmp = 0;
            scanf("%d", &nTmp);
            if (1 == nTmp)
            {
                lParam3 = 76;
            }
            else if (2 == nTmp)
            {
                lParam3 = 96;
            }
        }
        else if (DH_PTZ_LAMP_CONTROL == iterFind->m_iCmd)
        {
            // lParam1 表示开关控制
            printf("\t1:开启\n\t0:关闭\n\t 选择:");
            scanf("%d", &lParam1);
        }
        else if (DH_EXTPTZ_LEFTTOP <= iterFind->m_iCmd && iterFind->m_iCmd <=
DH_EXTPTZ_RIGHTDOWN)
        {
            // 垂直速度， 有效范围(1-8)
            lParam1 = 1;
            // 水平速度， 有效范围(1-8)
```

```
                lParam2 = 1;
            }
            else if (DH_EXTPTZ_ADDTOLOOP <= iterFind->m_iCmd && iterFind->m_iCmd <=
DH_EXTPTZ_DELFROMLOOP)
            {
                // lParam1 表示巡航路线
                printf("\t 巡航路线(%2d-%2d):",
stuPtzCapsInfo.wTourMin,stuPtzCapsInfo.wTourMax);
                scanf("%d", &lParam1);
                // lParam2 表示预置点号
                printf("\t 预置点号(%2d-%2d):",
stuPtzCapsInfo.wPresetMin,stuPtzCapsInfo.wPresetMax);
                scanf("%d", &lParam2);
            }
            else if (DH_EXTPTZ_CLOSELOOP == iterFind->m_iCmd)
            {
                // lParam1 表示巡航路线
                printf("\t 巡航路线(%2d-%2d):",
stuPtzCapsInfo.wTourMin,stuPtzCapsInfo.wTourMax);
                scanf("%d", &lParam1);
            }
            else if (DH_EXTPTZ_FASTGOTO == iterFind->m_iCmd)
            {
                // 水平坐标，有效范围(-8191 ~ 8191)
                lParam1 = 2000;
                // 水垂直坐标，有效范围(-8191 ~ 8191)
                lParam2 = 2000;
                // 变倍，有效范围(-16 ~ 16)
                lParam3 = 2;
            }
            else if (DH_EXTPTZ_EXACTGOTO == iterFind->m_iCmd)
            {
                // 水平坐标，有效范围，精度为能力集获取范围的 10 倍关系
                printf("\t 水平坐标(%2d-%2d):",
10*stuPtzCapsInfo.stuPtzMotionRange.nHorizontalAngleMin,
10*stuPtzCapsInfo.stuPtzMotionRange.nHorizontalAngleMax);
                scanf("%d", &lParam1);
                // 垂直坐标，有效范围，精度为能力集获取范围的 10 倍关系
                printf("\t 垂直坐标(%2d-%2d):",
10*stuPtzCapsInfo.stuPtzMotionRange.nVerticalAngleMin,
10*stuPtzCapsInfo.stuPtzMotionRange.nVerticalAngleMax);
                scanf("%d", &lParam2);
                // 变倍，有效范围（1 ~ 128）
                lParam3 = 2;
```

```
            }

            if (FALSE == CLIENT_DHPTZControlEx2(g_lLoginHandle, nChannelId,
iterFind->m_iCmd, lParam1, lParam2, lParam3, FALSE, pParam4))
            {
                printf("CLIENT_DHPTZControlEx2 Failed, nChoose[%x]!Last Error[%x]\n" , nChoose,
CLIENT_GetLastError());
            }
            cLastChoose = *iterFind;
        }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

int main()
{
    InitTest();

    RunTest();
```

```
    EndTest();

    return 0;
}

//**************************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
int GetIntInput(char *szPromt, int& nError)
{
    long int nGet = 0;
    char* pError = NULL;
    printf(szPromt);
    char szUserInput[32] = "";
    gets(szUserInput);
    nGet = strtol(szUserInput, &pError, 10);
```

```
    if ('\0' != *pError)
    {
        // 入参有误
        nError = -1;
    }
    else
    {
        nError = 0;
    }
    return nGet;
}
```

2.10 语音对讲

2.10.1 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。

语音对讲模式

语音对讲模式包括客户端模式和服务器模式两种，关于两种模式的详细流程请参见“2.10.3 流程说明”。

2.10.2 接口总览

表2-10 语音对讲接口说明

接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_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_SetDeviceMode	设置设备模式
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_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.10.3 流程说明

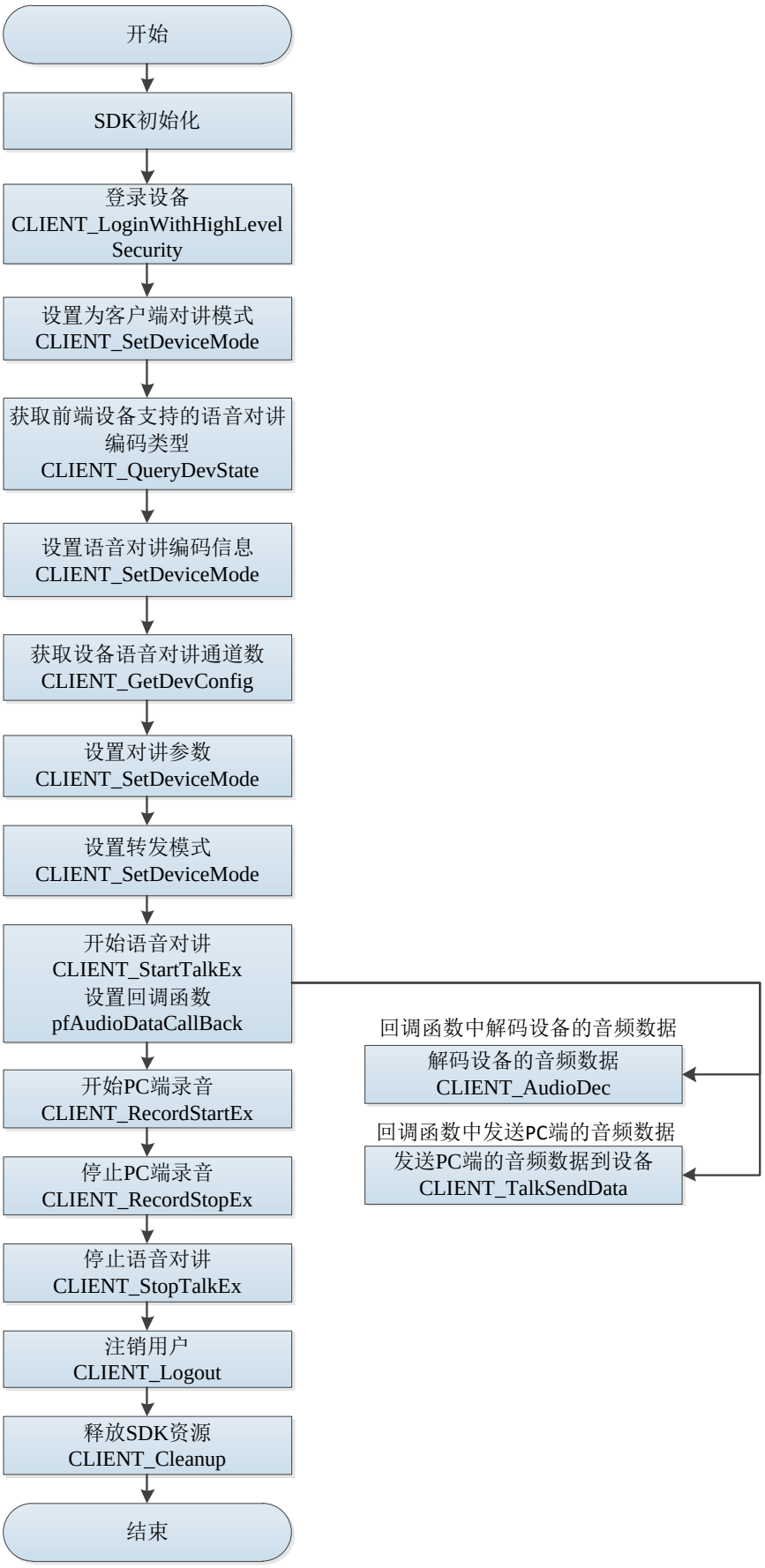
语音对讲包括以下两种模式：

- 客户端模式
  即 SDK 允许用户提供一个回调函数，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，会回调该函数。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，也可以将接收到的前端设备的音频数据进行解码播放。该模式只在 Windows 平台下有效。
- 服务器模式
  即 SDK 允许用户提供一个回调函数，当 SDK 接收到前端发送过来的音频数据时，会回调该函数。用户可在回调函数中保存接收到的前端设备的音频数据（用于后期业务功能使用，比如音频数据转发，调用第三方库解码播放音频数据等）。对于本地音频数据，用户可调用第三方库进行采集，然后调用 SDK 接口发送音频数据到设备端。

2.10.3.1 客户端模式

客户端模式流程如图 2-13 所示。

图2-13 客户端流程



流程说明

步骤1　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_SetDeviceMode 参数 emType 为 DH_TALK_CLIENT_MODE，设置对讲模式为客户端对讲模式。

步骤4 调用 CLIENT_SetDeviceMode 参数 emType 为 DH_TALK_ENCODE_TYPE，设置语音对讲编码信息。

步骤5 调用 CLIENT_SetDeviceMode 参数 emType 为 DH_TALK_SPEAK_PARAM，设置语音对讲参数。

步骤6 调用 CLIENT_SetDeviceMode 参数 emType 为 DH_TALK_TRANSFER_MODE，设置语音对讲转发模式。非转发模式，即本地 PC 与登录的设备之间实现语音对讲；转发模式，即本地 PC 与登录设备相应通道上连接的前端设备之间实现语音对讲。

步骤7 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8 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9 对讲功能使用完毕后，调用 CLIENT_RecordStopEx，停止 PC 端录音。

步骤10 调用 CLIENT_StopTalkEx，停止语音对讲。

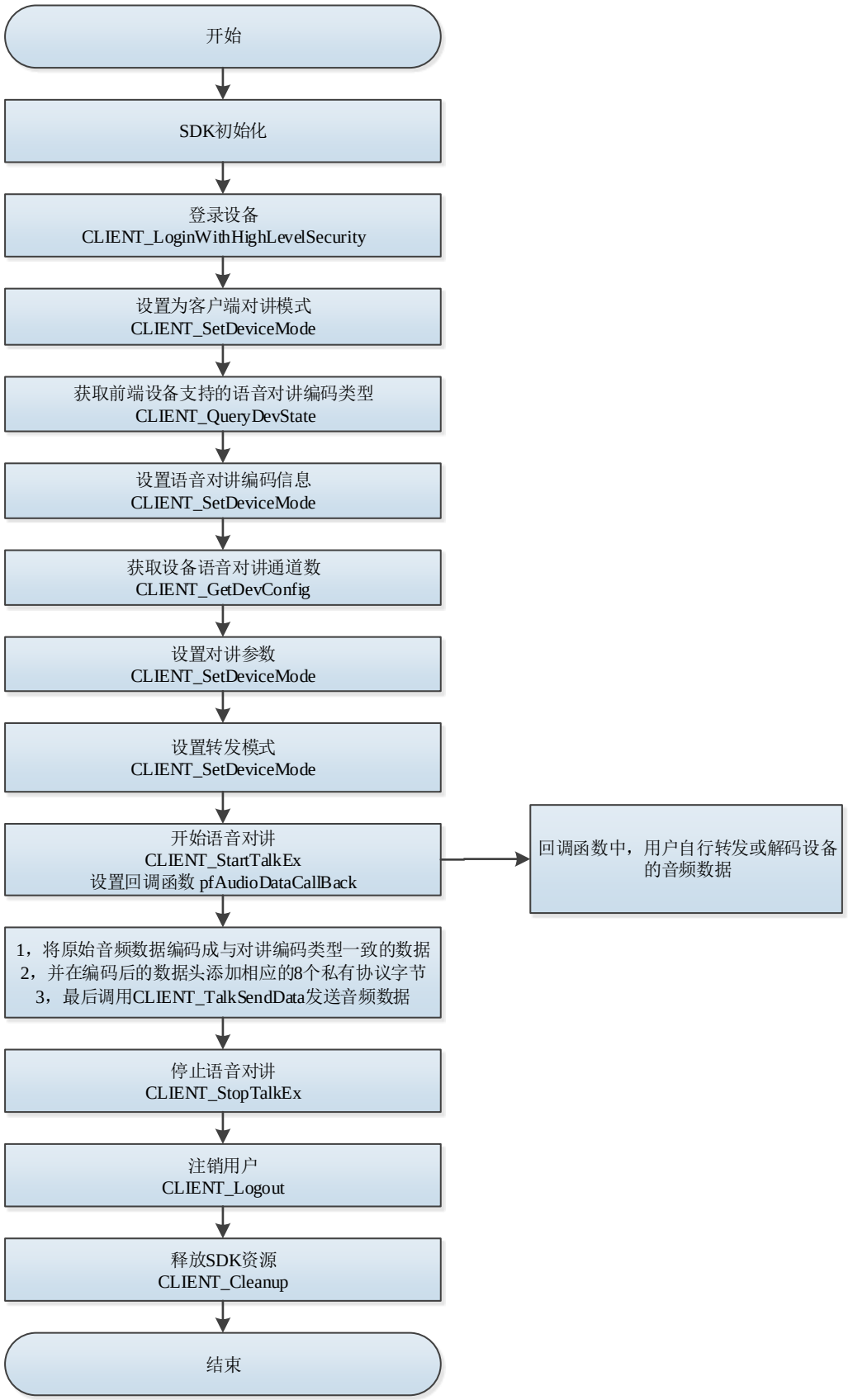
步骤11 调用 CLIENT_Logout，注销用户。

步骤12 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.10.3.2 服务器模式

服务器流程如图 2-14 所示。

图2-14 服务器流程



流程说明

步骤1　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_SetDeviceMode 参数 emType 为 DH_TALK_SERVER_MODE 将对讲模式设置为服务器对讲模式。

步骤4 调用 CLIENT_SetDeviceMode 参数 emType 为 DH_TALK_ENCODE_TYPE，设置语音对讲编码信息。

步骤5 调用 CLIENT_SetDeviceMode 参数 emType 为 DH_TALK_SPEAK_PARAM，设置语音对讲参数。

步骤6 调用 CLIENT_SetDeviceMode 参数 emType 为 DH_TALK_TRANSFER_MODE，设置语音对讲转发模式。非转发模式，即本地 PC 与登录的设备之间实现语音对讲；转发模式，即本地 PC 与登录设备相应通道上连接的前端设备之间实现语音对讲。

步骤7 调用 CLIENT_StartTalkEx，设置回调函数并开始语音对讲。在回调函数中，对于设备发过来的音频数据，用户自行处理，可以转发也可以解码播放。

步骤8 用户自行将原始音频数据编码成与对讲编码类型一致的数据，并在编码后的数据前添加 8 个相应的私有协议字节，最后调用 CLIENT_TalkSendData，发送音频数据。

步骤9 对讲功能使用完毕后，调用 CLIENT_StopTalkEx，停止语音对讲。

步骤10 调用 CLIENT_Logout，注销用户。

步骤11 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.10.4 示例代码

2.10.4.1 客户端模式

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"
#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lTalkHandle = 0L;
static BOOL g_bRecordFlag = FALSE;
static char g_szDevIp[32] = "192.168.1.10";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//************************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数。
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
```

```
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数。
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 语音对讲的音频数据回调函数
// 不建议在该回调函数中调用 SDK 接口，但该回调函数可以调用 CLIENT_TalkSendData 和 CLIENT_AudioDec SDK 接口
// 通过 CLIENT_StartTalkEx 中设置该回调函数，当收到本地 PC 端检测到的声卡数据，或者收到设备端发送过来的语音数据时，SDK 会调用该函数
void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag, DWORD dwUser);

//*************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }
    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;        // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);
```

```
    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与
CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

   NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头
文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误
码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" ,
g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp ,
g_nPort);
        }
        // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一
小段时间，具体等待时间因设备而异。
        Sleep(1000);
        printf("\n");
```

```
    }
}

void RunTest()
{

    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }

    // 设置为客户端对讲模式
    BOOL bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, DH_TALK_CLIENT_MODE,
NULL);
    if (FALSE == bSuccess)
    {
        printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n" ,
DH_TALK_CLIENT_MODE, CLIENT_GetLastError());
        return;
    }
    // 设置语音对讲编码信息
    DHDEV_TALKDECODE_INFO curTalkMode;
    curTalkMode = DH_TALK_PCM;
    bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, DH_TALK_ENCODE_TYPE,
&curTalkMode);
    if (FALSE == bSuccess)
    {
        printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n" ,
DH_TALK_ENCODE_TYPE, CLIENT_GetLastError());
        return;
    }
    // 设置对讲参数
    NET_SPEAK_PARAM stuSpeak = {sizeof(stuSpeak)};
    stuSpeak.nMode = 0; // 0：对讲（默认模式），1：喊话；从喊话切换到对讲要重新设置

    stuSpeak.nSpeakerChannel = 0; // 语音对讲通道号,默认为 0
    bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, DH_TALK_SPEAK_PARAM,
&stuSpeak);
    if (FALSE == bSuccess)
```

```
    {
        printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n" ,
DH_TALK_SPEAK_PARAM, CLIENT_GetLastError());
        return;
    }

    // 设置转发模式
    NET_TALK_TRANSFER_PARAM stuTransfer = {sizeof(stuTransfer)};
    stuTransfer.bTransfer = FALSE; // 由于是对登录设备对讲，关闭转发模式
    bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, DH_TALK_TRANSFER_MODE,
&stuTransfer);
    if (FALSE == bSuccess)
    {
        printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n" ,
DH_TALK_TRANSFER_MODE, CLIENT_GetLastError());
        return;
    }

    g_lTalkHandle = CLIENT_StartTalkEx(g_lLoginHandle, AudioDataCallBack, (DWORD)NULL);
    if(0 != g_lTalkHandle)
    {
        //   开始本地录音，如果只是从 DVR 到 PC 机的单向语音对讲可以不用调用该接口
        BOOL bSuccess = CLIENT_RecordStartEx(g_lLoginHandle);
        if(TRUE == bSuccess)
        {
            g_bRecordFlag = TRUE;
        }
        else
        {
            if (FALSE == CLIENT_StopTalkEx(g_lTalkHandle))
            {
                printf("CLIENT_StopTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
            }
            else
            {
                g_lTalkHandle = 0;
            }
        }
    }
    else
    {
        printf("CLIENT_StartTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
    }
```

```
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 停止本地录音
    if (TRUE == g_bRecordFlag)
    {
        if (!CLIENT_RecordStopEx(g_lLoginHandle))
        {
            printf("CLIENT_RecordStop Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_bRecordFlag = FALSE;
        }
    }
    // 停止语音对讲
    if (0 != g_lTalkHandle)
    {
        if(FALSE == CLIENT_StopTalkEx(g_lTalkHandle))
        {
            printf("CLIENT_StopTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lTalkHandle = 0;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
```

```
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }

    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//**********************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
```

```
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize,
BYTE byAudioFlag, DWORD dwUser)
{
    // 若多个对讲使用相同的数据回调函数，则用户可通过 lTalkHandle 进行一一对应
    if (g_lTalkHandle != lTalkHandle)
    {
        return;
    }

    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，调用 CLIENT_RecordStartEx 接
口后才会有相应数据过来
        LONG lSendLen = CLIENT_TalkSendData(lTalkHandle, pDataBuf, dwBufSize);
        if(lSendLen != (LONG)dwBufSize)
        {
            printf("CLIENT_TalkSendData Failed!Last Error[%x]\n" , CLIENT_GetLastError());
        }
    }
    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);
#ifdef _DEBUG
        FILE *stream;
        if( (stream = fopen("E:\\Talk.txt", "a+b")) != NULL )
        {
            int numwritten = fwrite( pDataBuf, sizeof( char ), dwBufSize, stream );
            fclose( stream );
        }
#endif
    }
}
```

2.10.4.2 服务器模式

```
#include <windows.h>
#include <stdio.h>
#include "dhplay.h"
```

```
#include "Alaw_encoder.h"
#include "dhnetsdk.h"

#pragma comment(lib , "dhplay.lib") // 第三方编解码库，示例中以编解码库为例
#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lTalkHandle = 0L;
static BOOL g_bOpenAudioRecord = FALSE;
static char g_szDevIp[32] = "192.168.1.10";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
static LLONG    g_nPlayPort = 0;
static DHDEV_TALKDECODE_INFO g_curTalkMode;
//*****************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数。
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用
该函数。
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser);

// 语音对讲的音频数据回调函数
// 在服务器模式下只会收到设备端发来的音频数据
// 不建议在该回调函数中调用 SDK 接口，但该回调函数可以调用 CLIENT_TalkSendData 和
CLIENT_AudioDec SDK 接口
// 通过 CLIENT_StartTalkEx 中设置该回调函数，当收到本地 PC 端检测到的声卡数据，或者收到
设备端发送过来的语音数据时，SDK 会调用该函数
void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize,
BYTE byAudioFlag, DWORD dwUser);

// PC 端音频编码发送回调函数
// pDataBuffer 为原始音频数据, DataLength 为有效数据长度
```

```
// 通过编解码库的 PLAY_OpenAudioRecord 接口设置，当检测到声卡数据后，编解码库会调用该函数
void CALLBACK AudioCallFunction(LPBYTE pDataBuffer, DWORD DataLength, void* pUser);

//*************************************************************************************
// 函数前向声明
// 该接口为调用编解码库实现语音对讲音频采集的示例，使用编解码库获取 PC 端原始音频码流
BOOL StartAudioRecord();
BOOL StopAudioRecord();

//*************************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }
    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;       // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
```

```
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头
文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误
码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" ,
g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp ,
g_nPort);
        }
        // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一
小段时间，具体等待时间因设备而异。
        Sleep(1000);
        printf("\n");
    }
}
```

```
void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }

    // 设置为服务器对讲模式
    BOOL bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, DH_TALK_SERVER_MODE,
NULL);
    if (FALSE == bSuccess)
    {
        printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n" ,
DH_TALK_SERVER_MODE, CLIENT_GetLastError());
        return;
    }
    // 设置语音对讲编码信息
    g_curTalkMode.encodeType = DH_TALK_PCM;
    bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, DH_TALK_ENCODE_TYPE,
&g_curTalkMode);
    if (FALSE == bSuccess)
    {
        printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n" ,
DH_TALK_ENCODE_TYPE, CLIENT_GetLastError());
        return;
    }
    // 设置对讲参数
    NET_SPEAK_PARAM stuSpeak = {sizeof(stuSpeak)};
    stuSpeak.nMode = 0; // 0：对讲（默认模式），1：喊话；从喊话切换到对讲要重新设置

        stuSpeak.nSpeakerChannel = 0; // 语音对讲通道号，默认为 0
    bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, DH_TALK_SPEAK_PARAM,
&stuSpeak);
    if (FALSE == bSuccess)
    {
        printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n" ,
DH_TALK_SPEAK_PARAM, CLIENT_GetLastError());
        return;
    }
```

```
    // 设置转发模式
    NET_TALK_TRANSFER_PARAM stuTransfer = {sizeof(stuTransfer)};
    stuTransfer.bTransfer = FALSE; // 由于是对登录设备对讲，关闭转发模式
    bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, DH_TALK_TRANSFER_MODE,
&stuTransfer);
    if (FALSE == bSuccess)
    {
        printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n" ,
DH_TALK_TRANSFER_MODE, CLIENT_GetLastError());
        return;
    }

    g_lTalkHandle = CLIENT_StartTalkEx(g_lLoginHandle, AudioDataCallBack, (DWORD)NULL);
    if(0 != g_lTalkHandle)
    {
        bSuccess = StartAudioRecord();
        if(TRUE == bSuccess)
        {
            g_bOpenAudioRecord = TRUE;
        }
        else
        {
            printf("StartAudioRecord Failed!\n");
            CLIENT_StopTalkEx(g_lTalkHandle);
            g_lTalkHandle = 0;
        }
    }
    else
    {
        printf("CLIENT_StartTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 清理编解码库对讲资源
    if(TRUE == g_bOpenAudioRecord)
    {
        if (TRUE == StopAudioRecord())
        {
```

```
            g_bOpenAudioRecord = FALSE;
        }
    }

    // 停止语音对讲
    if (0 != g_lTalkHandle)
    {
        if(FALSE == CLIENT_StopTalkEx(g_lTalkHandle))
        {
            printf("CLIENT_StopTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lTalkHandle = 0;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }

    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
```

```
    return 0;
}

//*************************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize,
BYTE byAudioFlag, DWORD dwUser)
{
    // 若多个对讲使用相同的数据回调函数，则用户可通过 lTalkHandle 进行一一对应
    if (g_lTalkHandle != lTalkHandle)
    {
        return;
    }
```

```
    if(1 == byAudioFlag)
    {
        // 此处对于设备发过来的音频数据，用户自行处理，可以转发也可以解码播放
        // 下面是通过编解码库实现的示例代码
        PLAY_GetFreePort(&g_nPlayPort);
        //For PCM format withour header , please add 128.
        if (g_curTalkMode.encodeType == DH_TALK_DEFAULT)
        {
            for (unsigned int i = 0; i < dwBufSize; i++)
            {
                pDataBuf[i] += (char)128;
            }
        }

        //You can use PLAY SDK to decode to get PCM and then encode to other formats if
you to get a uniform formats.
        PLAY_InputData(g_nPlayPort,(BYTE *)pDataBuf,dwBufSize);
#ifdef _DEBUG
        FILE *stream;
        if( (stream = fopen("E:\\Talk.txt", "a+b")) != NULL )
        {
            int numwritten = fwrite( pDataBuf, sizeof( char ), dwBufSize, stream );
            fclose( stream );
        }
#endif
    }
}

void CALLBACK AudioCallFunction(LPBYTE pDataBuffer, DWORD DataLength, void* pUser)
{
    char* pCbData = NULL;
    pCbData = new char[102400];
    if (NULL == pCbData)
    {
        return;
    }
    int   iCbLen = 0;
    // 对讲码流前 8 个字节为私有协议字节，8 字节以后为相应的对讲编码类型的音频数据
    // 示例代码中提供 PCM，g711a，g711u 编码对应的前 8 个私有协议字节的实现方式
    if (g_curTalkMode.encodeType == DH_TALK_DEFAULT || g_curTalkMode.encodeType ==
DH_TALK_PCM)
    {
        if (g_curTalkMode.nAudioBit == 8)
```

```
        {
            for(unsigned int j = 0 ; j < DataLength; j++)
            {
                *(pDataBuffer + j) += 128;
            }
        }

        pCbData[0]=0x00;
        pCbData[1]=0x00;
        pCbData[2]=0x01;
        pCbData[3]=0xF0;

        pCbData[4]=g_curTalkMode.nAudioBit==8?0x07:0x0C;
        if( 8000 == g_curTalkMode.dwSampleRate )
        {
            pCbData[5]=0x02;//8k
        }
        else if(16000 == g_curTalkMode.dwSampleRate)
        {
            pCbData[5] = 0x04;
        }
        else if(48000 == g_curTalkMode.dwSampleRate)
        {
            pCbData[5] = 0x09;
        }

        *(DWORD*)(pCbData+6)=DataLength;
        memcpy(pCbData+8, pDataBuffer, DataLength);

        iCbLen = 8+DataLength;
    }
    else if (g_curTalkMode.encodeType == DH_TALK_G711a)
    {
        // 将原始音频数据编码为 g711a
        if (g711a_Encode((char*)pDataBuffer, pCbData+8, DataLength, &iCbLen) != 1)
        {
            goto end;
        }

        //Private bit stream format frame head
        pCbData[0]=0x00;
        pCbData[1]=0x00;
        pCbData[2]=0x01;
```

```
        pCbData[3]=0xF0;

        pCbData[4]=0x0E; //G711A

        if( 8000 == g_curTalkMode.dwSampleRate )
        {
            pCbData[5]=0x02;//8k
        }
        else if(16000 == g_curTalkMode.dwSampleRate)
        {
            pCbData[5] = 0x04;
        }
        else if(48000 == g_curTalkMode.dwSampleRate)
        {
            pCbData[5] = 0x09;
        }

        pCbData[6]=BYTE(iCbLen&0xff);
        pCbData[7]=BYTE(iCbLen>>8);

        iCbLen += 8;
    }
    else if (g_curTalkMode.encodeType == DH_TALK_G711u)
    {
         // 将原始音频数据编码为 g711u
        if (g711u_Encode((char*)pDataBuffer, pCbData+8, DataLength, &iCbLen) != 1)
        {
            goto end;
        }

        //Private bit stream format frame head
        pCbData[0]=0x00;
        pCbData[1]=0x00;
        pCbData[2]=0x01;
        pCbData[3]=0xF0;

        pCbData[4]=0x0A; //G711u
        if( 8000 == g_curTalkMode.dwSampleRate )
        {
            pCbData[5]=0x02;//8k
        }
        else if(16000 == g_curTalkMode.dwSampleRate)
        {
```

```
            pCbData[5] = 0x04;
        }
        else if(48000 == g_curTalkMode.dwSampleRate)
        {
            pCbData[5] = 0x09;
        }

        pCbData[6]=BYTE(iCbLen&0xff);
        pCbData[7]=BYTE(iCbLen>>8);

        iCbLen += 8;
    }
    else
    {
        goto end;
    }

    // Send the data from the PC to DVR
    CLIENT_TalkSendData(g_lTalkHandle, (char *)pCbData, iCbLen);

end:
    if (pCbData != NULL)
    {
        delete[] pCbData;
    }
}

//*************************************************************************************
BOOL StartAudioRecord()
{
    // 此为编解码库特性，不细究
    PLAY_GetFreePort(&g_nPlayPort);

    //    Then specify frame length
    int nFrameLength = 1024;
    switch(g_curTalkMode.encodeType)
    {
    case DH_TALK_DEFAULT:
    case DH_TALK_PCM:
        nFrameLength = 1024;
        break;
    case DH_TALK_G711a:
        nFrameLength = 1280;
```

```
        break;
    case DH_TALK_AMR:
        nFrameLength = 320;
        break;
    case DH_TALK_G711u:
        nFrameLength = 320;
        break;
    case DH_TALK_G726:
        nFrameLength = 320;
        break;
    case DH_TALK_AAC:
        nFrameLength = 1024;
    default:
        break;
    }

    if (g_curTalkMode.dwSampleRate == 48000)//如果采样率 48K，更新音频长度
    {
        nFrameLength = 48*40*2; //  采样率*40*2
    }

    BOOL bRet = FALSE;

    //    Then call PLAYSDK library to begin recording audio
    BOOL bOpenRet = PLAY_OpenStream(g_nPlayPort,0,0,1024*900);
    if(bOpenRet)
    {
        BOOL bPlayRet = PLAY_Play(g_nPlayPort,0);
        if(bPlayRet)
        {
            PLAY_PlaySoundShare(g_nPlayPort);
            BOOL bSuccess =
PLAY_OpenAudioRecord(AudioCallFunction,g_curTalkMode.nAudioBit,
g_curTalkMode.dwSampleRate,nFrameLength,0,NULL);
            if(bSuccess)
            {
                bRet = TRUE;
            }
            else
            {
                PLAY_StopSoundShare(g_nPlayPort);
                PLAY_Stop(g_nPlayPort);
                PLAY_CloseStream(g_nPlayPort);
```

```
            }
        }
        else
        {
            PLAY_CloseStream(g_nPlayPort);
        }
    }

    return bRet;
}

BOOL StopAudioRecord()
{
    // 此为编解码库特性，不细究
    BOOL bSuccess = PLAY_CloseAudioRecord();
    if(TRUE == bSuccess)
    {
        PLAY_Stop(g_nPlayPort);
        PLAY_StopSoundShare(g_nPlayPort);
        PLAY_CloseStream(g_nPlayPort);
    }
    else
    {
        printf("PLAY_CloseAudioRecord Failed!\n");
    }

    return bSuccess;
}
```

2.11 视频抓图

2.11.1 简介

视频抓图，可以从视频中获取图片，也可以从设备中获取图片，用于上层用户实现平台开发需求。

从设备中获取图片：用户调用 SDK 接口发送抓图命令给设备，设备在实时预览中抓取当前画面，并发送给 SDK，SDK 将接收到的图片数据返回给用户。

2.11.2 接口总览

表2-11 视频抓图接口说明

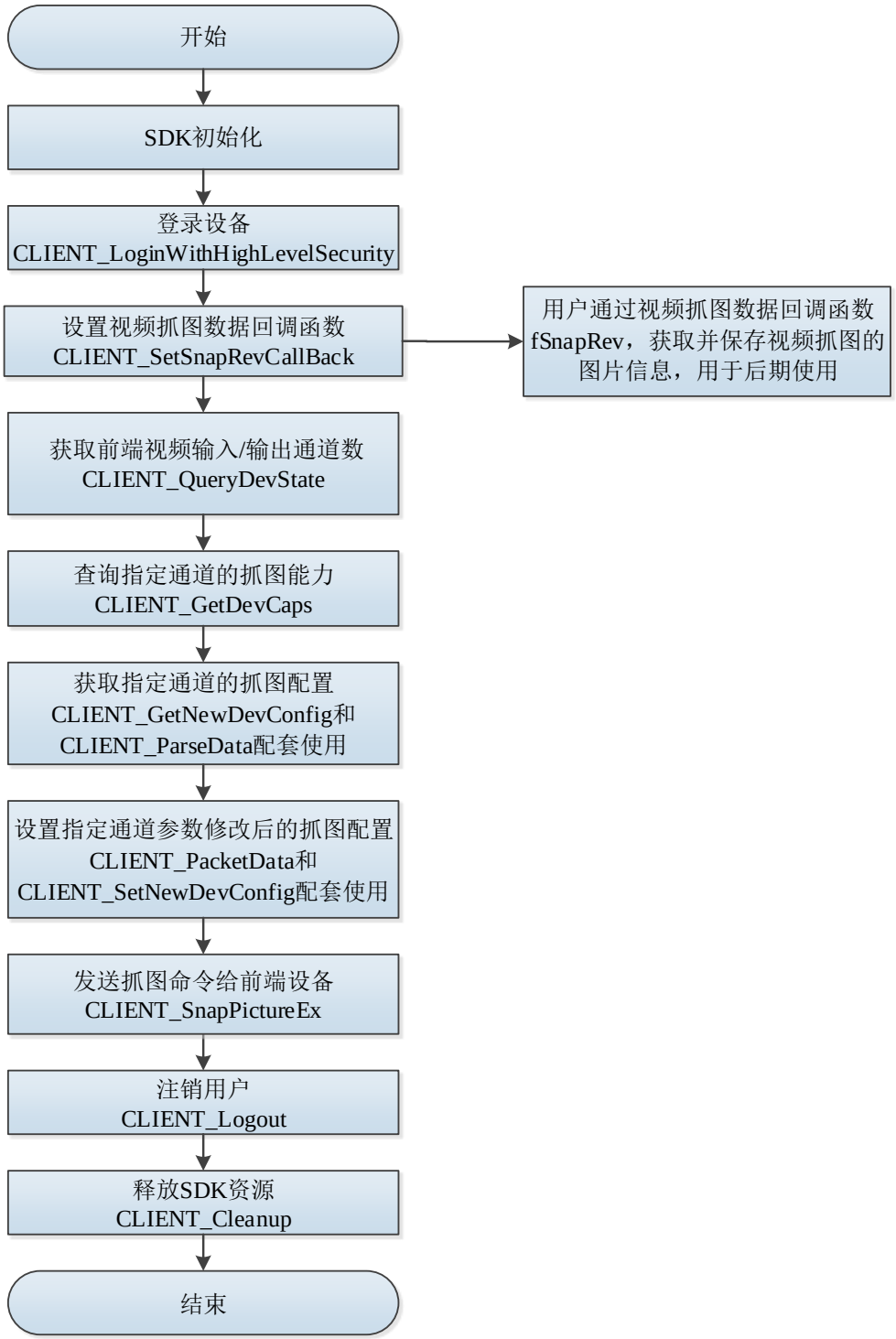
接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口

接口	接口说明
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_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_SetSnapRevCallBack	设置视频抓图数据回调函数
CLIENT_SnapPictureEx	抓图请求扩展接口
CLIENT_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.11.3 流程说明

视频抓图流程如图 2-15 所示。

图2-15 视频抓图流程



流程说明

步骤1　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_SetSnapRevCallBack 设置抓图回调函数，当 SDK 收到设备端发送过来的抓图数据时，会调用 fSnapRev 回调函数回调图片信息及图片数据给用户。

步骤4　调用 CLIENT_SnapPictureEx 发送抓图命令给前端设备，在 fSnapRev 回调函数中等待设备回复的图片信息。

步骤5　调用 CLIENT_Logout，注销用户。

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

2.11.4 示例代码

```
#include <windows.h>
#include <stdio.h>
#include <time.h>
#include "dhnetsdk.h"
#include "dhconfigsdk.h"

#pragma comment(lib, "dhnetsdk.lib")
#pragma comment(lib, "dhconfigsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static char g_szDevIp[32] = "192.168.1.10";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
static short g_nCmdSerial = 0;  // 抓图序列号

//************************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数。
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用
该函数。
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser);

// 抓图回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetSnapRevCallBack 设置该回调函数，当前端设备有抓图数据发送过来时，SDK
会调用该函数
void CALLBACK SnapRev(LLONG lLoginID, BYTE *pBuf, UINT RevLen, UINT EncodeType, DWORD
CmdSerial, LDWORD dwUser);

//************************************************************************************
// 常用函数声明
```

```
// 获取输入的整形
int GetIntInput(char *szPromt, int& nError);

// 获取输入的字符串
void GetStringInput(const char *szPromt , char *szBuffer);

//*************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;       // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
```

```
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一小段时间，具体等待时间因设备而异。
        Sleep(1000);
        printf("\n");
    }
}


void RunTest()
```

```
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }
      // 设置抓图回调函数
CLIENT_SetSnapRevCallBack(SnapRev, NULL);
      //事例中默认通道 ID 为 0、抓图模式为抓一幅图,用户可根据实际情况自行选择
        int    nChannelId = 0;
        int     nSnapType = 0;// 抓图模式：-1:表示停止抓图, 0：表示请求一帧, 1：表示定时发送请求, 2：表示连续请求
        // 发送抓图命令给前端设备
        SNAP_PARAMS stuSnapParams;
        stuSnapParams.Channel = nChannelId;
        stuSnapParams.mode = nSnapType;
        stuSnapParams.CmdSerial = ++g_nCmdSerial; // 请求序列号，有效值范围 0~65535，超过范围会被截断为 unsigned short
        if (FALSE == CLIENT_SnapPictureEx(g_lLoginHandle, &stuSnapParams))
        {
            printf("CLIENT_SnapPictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
            return;
        }
        else
        {
            printf("CLIENT_SnapPictureEx succ\n");
        }

        GetStringInput("'q': 退出; 'c': 继续\n", szUserChoose);
    }while('q' != szUserChoose[0]);

    return;
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
```

```
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }

    exit(0);
}

int main()
{
    InitTest();

    RunTest();

    EndTest();

    return 0;
}

//************************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
```

```
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK SnapRev(LLONG lLoginID, BYTE *pBuf, UINT RevLen, UINT EncodeType, DWORD
CmdSerial, LDWORD dwUser)
{
    printf("[SnapRev] -- receive data!\n");
    if(lLoginID == g_lLoginHandle)
    {
        if (NULL != pBuf && RevLen > 0)
        {
            char szPicturePath[256] = "";
            time_t stuTime;
            time(&stuTime);
            char szTmpTime[128] = "";
            strftime(szTmpTime, sizeof(szTmpTime) - 1, "%y%m%d_%H%M%S",
gmtime(&stuTime));
            _snprintf(szPicturePath, sizeof(szPicturePath)-1, "%d_%s.jpg", CmdSerial,
szTmpTime);
            FILE* pFile = fopen(szPicturePath, "wb");
            if (NULL == pFile)
            {
                return;
            }
```

```
                int nWrite = 0;
                while(nWrite != RevLen)
                {
                    nWrite += fwrite(pBuf + nWrite, 1, RevLen - nWrite, pFile);
                }

                fclose(pFile);
            }

        }
}

//*****************************************************************************
// 常用函数定义
int GetIntInput(char *szPromt, int& nError)
{
    long int nGet = 0;
    char* pError = NULL;
    printf(szPromt);
    char szUserInput[32] = "";
    gets(szUserInput);
    nGet = strtol(szUserInput, &pError, 10);
    if ('\0' != *pError)
    {
        // 入参有误
        nError = -1;
    }
    else
    {
        nError = 0;
    }

    return nGet;
}

void GetStringInput(const char *szPromt , char *szBuffer)
{
    printf(szPromt);
    gets(szBuffer);
}
```

2.12 报警上报

2.12.1 简介

报警上报，即前端设备在检测到事先规定的特殊事件发生时，发送报警到平台端告知平台。平台可以接收到设备上传的外部报警、视频信号丢失报警、遮挡报警和动态检测报警等信息。

报警上报实现方式为 SDK 主动连接设备，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。

2.12.2 接口总览

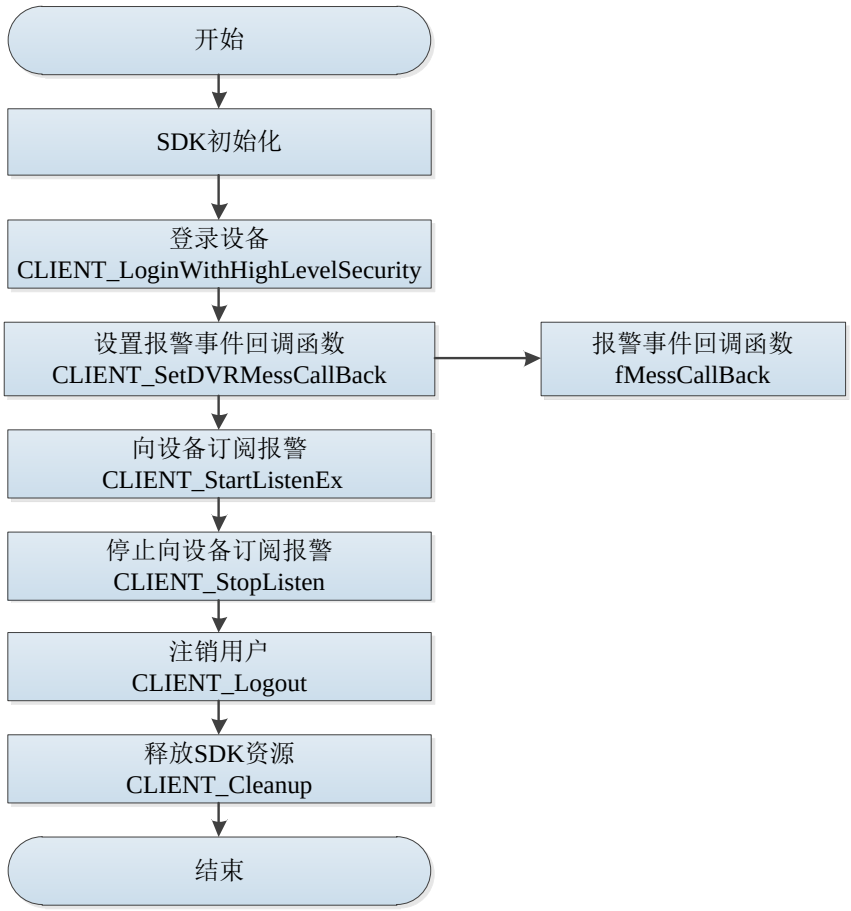
表2-12 报警上报接口说明

接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 CLIENT_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_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
CLIENT_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.12.3 流程说明

报警上报流程如图 2-16 所示。

图2-16 报警上报流程



流程说明

步骤1　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_SetDVRMessCallBack，设置报警事件回调函数，该接口需在报警订阅之前调用。
步骤4　调用 CLIENT_StartListenEx，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。
步骤5　报警上报功能使用完毕后，调用 CLIENT_StopListen，停止向设备订阅报警。
步骤6　调用 CLIENT_Logout，退出设备。
步骤7　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.12.4 示例代码

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static char g_szDevIp[32] = "192.168.1.10";
static WORD g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
```

```
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
static BOOL g_bStartListenFlag = FALSE;

//*************************************************************************************
// 常用回调集合声明

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数。
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数。
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 报警事件回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetDVRMessCallBack 设置该回调函数，当接收到设备上报的报警事件时，SDK 会调用该函数
BOOL CALLBACK MessCallBack(LONG lCommand, LLONG lLoginID, char *pBuf, DWORD dwBufLen, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

//*************************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }
    // 获取 SDK 版本信息
    // 此操作为可选操作
```

```
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;          // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

   NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
```

```
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23) // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一小段时间，具体等待时间因设备而异。
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }
    // 设置报警事件回调函数
    CLIENT_SetDVRMessCallBack(MessCallBack , NULL);

    // 向设备订阅报警
    if( TRUE == CLIENT_StartListenEx(g_lLoginHandle))
    {
        g_bStartListenFlag = TRUE;
        printf("CLIENT_StartListenEx Success!\nJust Wait Event....\n");
    }
    else
    {
        printf("CLIENT_StartListenEx Failed!Last Error[%x]\n" , CLIENT_GetLastError());
    }
}
```

```
void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // 停止向设备订阅报警
    if (TRUE == g_bStartListenFlag)
    {
        if (FALSE == CLIENT_StopListen(g_lLoginHandle))
        {
            printf("CLIENT_StopListen Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_bStartListenFlag = FALSE;
        }
    }
    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }
    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }

    return;
}

int main()
{
    InitTest();
    RunTest();
```

```
    EndTest();
    return 0;
}

//**************************************************************************
// 常用回调集合定义

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

BOOL CALLBACK MessCallBack(LONG lCommand, LLONG lLoginID, char *pBuf, DWORD
dwBufLen, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    printf("[MessCallBack] -- Get Event IP[%s] , port[%d]\n" , pchDVRIP , nDVRPort);

    // demo 只列出部分报警的处理方式，用户根据自己需求处理对应的报警事件信息，具体请
参见 dhnetsdk.h 头文件相应事件的使用说明
    switch(lCommand)
```

```
    {
    case DH_ALARM_ALARM_EX:
        {
            printf("\n 外部报警\n");
            if (NULL != pBuf)
            {
                BYTE* pInfo = (BYTE*)pBuf;
                for(unsigned int i = 0; i < dwBufLen/sizeof(BYTE); ++i)
                {
                    printf("nChannelID = [%2d], state = [%d]\n", i, *(pInfo + i));
                }
            }
        }
        break;
    case DH_MOTION_ALARM_EX:
        {
            printf("\n 动态检测报警\n");
            if (NULL != pBuf)
            {
                BYTE* pInfo = (BYTE*)pBuf;
                for(unsigned int i = 0; i < dwBufLen/sizeof(BYTE); ++i)
                {
                    printf("nChannelID = [%2d], state = [%d]\n", i, *(pInfo + i));
                }
            }
        }
        break;
    case DH_ALARM_ALARM_EX_REMOTE:
        {
            printf("\n 远程外部报警\n");
            if (NULL != pBuf)
            {
                ALARM_REMOTE_ALARM_INFO* pInfo = (ALARM_REMOTE_ALARM_INFO
*)pBuf;
                printf("nChannelID = %d\n" , pInfo->nChannelID);
                printf("nState = %d\n" , pInfo->nState);
            }
        }
        break;
    case DH_ALARM_ACCESS_CTL_EVENT:
        {
            printf("\n 门禁事件\n");
            if (NULL != pBuf)
```

```
            {
                ALARM_ACCESS_CTL_EVENT_INFO* pInfo =
(ALARM_ACCESS_CTL_EVENT_INFO *)pBuf;
                printf("开门方式  = %d\n" , pInfo->emOpenMethod);
                printf("卡号  = [%s]\n" , pInfo->szCardNo);
            }
        }
        break;
    default:
        printf("\n[MessCallBack] --  其他报警  Get lCommand = 0x%x\n" , lCommand);
        break;
    }
    return TRUE;
}
```

2.13 设备搜索

2.13.1 简介

设备搜索功能主要用于协助用户获取网络上的设备信息。设备搜索功能可与登录功能配合使用，通过设备搜索功能发现相关的设备，再通过登录功能登录设备。

设备搜索根据是否跨网段可分为以下两种：

- 异步同网段设备搜索

搜索当前操作所在网段内的设备信息。

- 同步跨网段设备搜索

根据用户设置的网段信息，搜索相应网段内的设备信息。

2.13.2 接口总览

表2-13 设备搜索接口说明

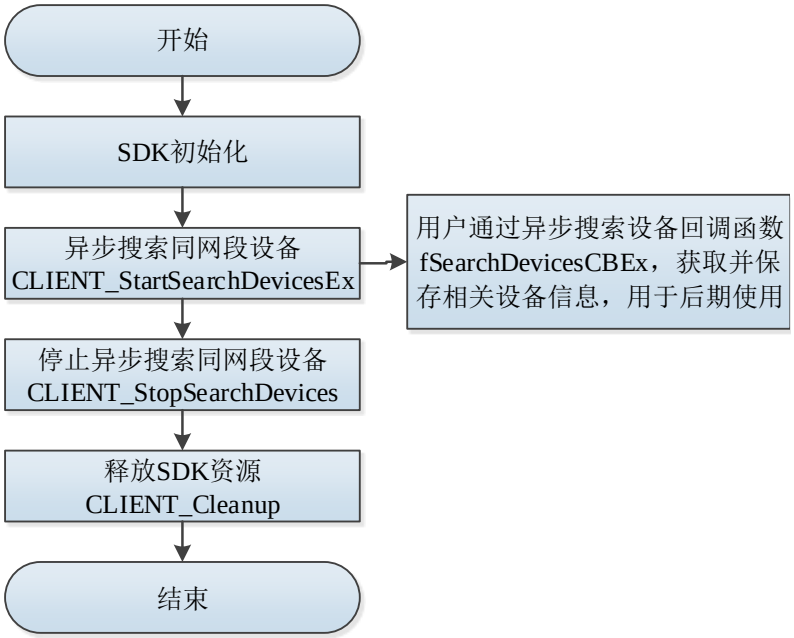
接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_StartSearchDevicesEx	异步搜索同网段内 IPC、NVS 等设备
CLIENT_StopSearchDevices	停止异步搜索同网段内 IPC、NVS 等设备
CLIENT_SearchDevicesByIPs	同步跨网段搜索设备
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.13.3 流程说明

2.13.3.1 异步搜索同网段设备

异步搜索同网段设备流程，如图 2-17 所示。

图2-17 异步搜索同网段设备流程



流程说明

步骤1　完成 SDK 初始化流程。

步骤2　调用 CLIENT_StartSearchDevicesEx，异步搜索同网段内的设备，用户通过该接口设置的 fSearchDevicesCB 类型的回调函数获取搜索到的设备信息（该搜索没有超时时间，需要用户调用 CLIENT_StopSearchDevices 接口停止搜索）。

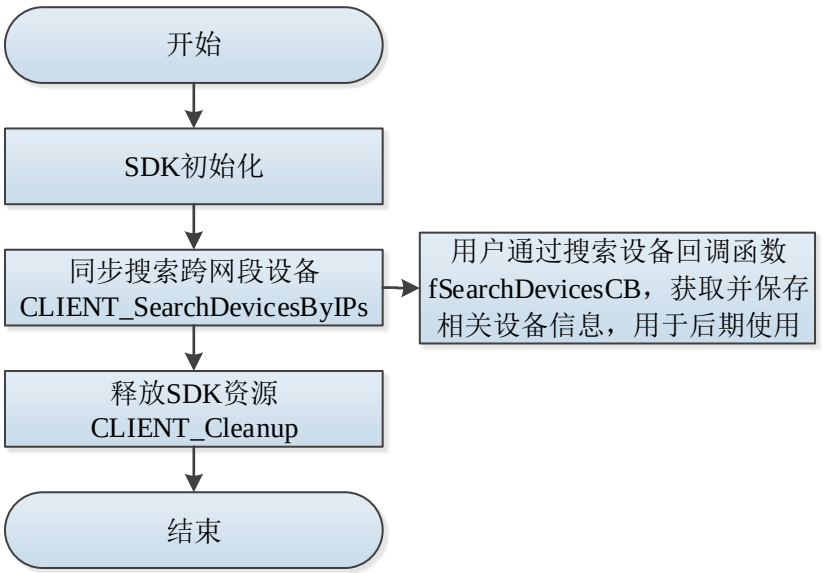
步骤3　调用 CLIENT_StopSearchDevices，停止异步搜索同网段内的设备。

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

2.13.3.2 同步搜索跨网段设备

同步搜索跨网段设备流程，如图 2-18 所示。

图2-18 同步搜索跨网段设备流程



流程说明

步骤1　调用 CLIENT_Init 初始化 SDK。

步骤2　调用 CLIENT_SearchDevicesByIPs，同步搜索跨网段内的设备，用户通过该接口设置的

fSearchDevicesCB 类型的回调函数获取搜索到的设备信息，当跨网段内的设备搜索完毕或达到该接口设置的超时时间，接口才会返回，用户需要根据自身网络情况决定超时时间。

步骤3　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

2.13.4 示例代码

2.13.4.1 异步搜索同网段设备

```
#include <windows.h>
#include <stdio.h>
#include <vector>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lSearchHandle = 0L;
static CRITICAL_SECTION g_mDeviceListLock;                    // 设备列表操作锁
static std::vector<DEVICE_NET_INFO_EX> g_lDeviceVec;      // 设备列表

//*************************************************************************************
//常用回调集合

// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 异步搜索设备回调
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_StartSearchDevicesEx/CLIENT_SearchDevicesByIPs 设置该回调函数，当搜索到设备时，SDK 会调用该函数
void CALLBACK cbSearchDevicesEx(LLONG lSearchHandle,DEVICE_NET_INFO_EX2 *pDevNetInfo, void* pUserData);
```

```
//*************************************************************************************
void InitTest()
{
    // 初始化线程锁
    InitializeCriticalSection(&g_mDeviceListLock);

    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }
    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

    // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;       // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);
}

void RunTest()
{
```

```
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
// 开始异步搜索同一局域网设备
NET_IN_STARTSERACH_DEVICE pInBuf = { 0 };
    NET_OUT_STARTSERACH_DEVICE pOutBuf = { 0 };
    LLONG seachHandle = 0;
pInBuf.dwSize = sizeof(NET_IN_STARTSERACH_DEVICE);
    pInBuf.cbSearchDevices = cbSearchDevicesEx;
pInBuf.pUserData = this;
int nMaxCopyLen = MAX_LOCAL_IP_LEN - 1;
    strncpy(pInBuf.szLocalIp, "192.168.1.10", sizeof(pInBuf.szLocalIp) - 1);
    pOutBuf.dwSize = sizeof(NET_OUT_STARTSERACH_DEVICE);

    seachHandle = CLIENT_StartSearchDevicesEx(&pInBuf, &pOutBuf);
    if (NULL == seachHandle)
    {
        printf("CLIENT_StartSearchDevicesEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
        return;
    }
    int nIndex = 0;
    int nSearchTime = 0;
    int nSearchLimit = 10;// 搜索持续十秒，用户可根据自己网络环境修改该值
    Sleep(nSearchLimit * 1000);
    EnterCriticalSection(&g_mDeviceListLock);
    for (std::vector<DEVICE_NET_INFO_EX>::iterator iter = g_lDeviceVec.begin(); iter !=
g_lDeviceVec.end(); ++iter)
    {
        printf("\n********** find device **********\n");
        printf("nIndex[%d]\n", ++nIndex);
        printf("iIPVersion[%d]\n", iter->iIPVersion);
        printf("szIP[%s]\n", iter->szIP);
        printf("nPort[%d]\n", iter->nPort);
    }
    g_lDeviceVec.clear();
    LeaveCriticalSection(&g_mDeviceListLock);
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
```

```
    // 清理线程锁资源
    DeleteCriticalSection(&g_mDeviceListLock);

    // 停止异步搜索同一局域网设备
    if (NULL != g_lSearchHandle)
    {
        if (FALSE == CLIENT_StopSearchDevices(g_lSearchHandle))
        {
            printf("CLIENT_StopSearchDevices Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
    }

    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}

int main()
{
    InitTest();

    RunTest();

    EndTest();

    return 0;
}

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
```

```
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
	printf("Call HaveReConnect\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}
void CALLBACK cbSearchDevicesEx(LLONG lSearchHandle,DEVICE_NET_INFO_EX2
*pDevNetInfo, void* pUserData)
{
if(pDevNetInfo != NULL)
{
CDevInitDlg *dlg = (CDevInitDlg *)pUserData;
DEVICE_NET_INFO_EX2 *pData = NEW DEVICE_NET_INFO_EX2;
memcpy(pData, pDevNetInfo, sizeof(DEVICE_NET_INFO_EX2));
LONG bIsUnicast = dlg->m_IsUnicast;
}
}
```

2.13.4.2 同步搜索跨网段设备

```
#include <windows.h>
#include <stdio.h>
#include <vector>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

BOOL g_bNetSDKInitFlag = FALSE;
std::vector<DEVICE_NET_INFO_EX> g_lDeviceVec;	// 设备列表

// ************************************* 获取本地 IP 接口
std::string GetLocalIpAddress();

//*****************************************************************************
//常用回调集合声明
```

```
// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用该函数
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// 搜索设备回调
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_StartSearchDevicesEx/CLIENT_SearchDevicesByIPs 设置该回调函数，当搜索到设备时，SDK 会调用该函数
void CALLBACK SearchDevicesCB(DEVICE_NET_INFO_EX *pDevNetInfo, void* pUserData);
//*****************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);
```

```
    // 设置连接设备超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 超时时间设置为 5 s
    int nTryTimes = 3;       // 若出现超时，尝试登录 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与
CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
    // 开始同步搜索跨网段设备
    char szLocalIp[64] = "";
    strncpy(szLocalIp, GetLocalIpAddress().c_str(), sizeof(szLocalIp) - 1);

    DEVICE_IP_SEARCH_INFO stuTmp = {sizeof(stuTmp)};
    stuTmp.nIpNum = 256;// 搜索的 ip 地址有效个数
    for (unsigned int i = 0; i < stuTmp.nIpNum; ++i)
    {
        // 用户需保证 ip 地址的有效性
        _snprintf(stuTmp.szIP[i], sizeof(stuTmp.szIP[i]) - 1, "192.168.1.%d", i);
    }

    DWORD dwWaitTime = 5000;
    // 用户需要注意，该接口会等到超时时间到时才返回，用户需要根据自身网络情况决定超时
时间
    if (FALSE == CLIENT_SearchDevicesByIPs(&stuTmp, SearchDevicesCB,
(LDWORD)&g_lDeviceVec, szLocalIp, dwWaitTime))
    {
        printf("CLIENT_SearchDevicesByIPs Failed!Last Error[%x]\n", CLIENT_GetLastError());
        return;
    }
```

```
    int nIndex = 0;
    for (std::vector<DEVICE_NET_INFO_EX>::iterator iter = g_lDeviceVec.begin(); iter !=
g_lDeviceVec.end(); ++iter)
    {
        printf("\n********** find device **********\n");
        printf("nIndex[%d]\n", ++nIndex);
        printf("iIPVersion[%d]\n", iter->iIPVersion);
        printf("szIP[%s]\n", iter->szIP);
        printf("nPort[%d]\n", iter->nPort);
    }
    g_lDeviceVec.clear();
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();

    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}

int main()
{
    InitTest();

    RunTest();

    EndTest();

    return 0;
}

//**************************************************************************
//常用回调集合定义
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
    printf("Call DisConnectFunc\n");
```

```
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK SearchDevicesCB(DEVICE_NET_INFO_EX *pDevNetInfo, void* pUserData)
{
    if ((NULL == pDevNetInfo) || (NULL == pUserData))
    {
        printf("warning param is null\n");
        return;
    }

    std::vector<DEVICE_NET_INFO_EX>* pDeviceList =
(std::vector<DEVICE_NET_INFO_EX>*)pUserData;
    pDeviceList->push_back(*pDevNetInfo);
    return;
}



// ************************************  获取本地 IP 接口
std::string GetLocalIpAddress()
{
    WSADATA wsaData;
```

```
    if (0 != WSAStartup(MAKEWORD(2,2), &wsaData))
    {
        return "";
    }

    char local[255] = "";
    gethostname(local, sizeof(local));
    hostent* ph = gethostbyname(local);
    if (NULL == ph)
    {
        return "";
    }

    in_addr addr;
    memcpy(&addr, ph->h_addr_list[0], sizeof(in_addr));

    std::string localIP(inet_ntoa(addr));
    WSACleanup();
    return localIP;
}
```

2.14 智能事件上报与抓图

2.14.1 简介

智能事件上报，是指设备端通过对实时预览码流的智能分析，根据用户在设备端配置的智能事件触发规则来判断是否需要上报事件以及是否携带图片给用户。智能事件包括场景变化、穿越警戒线、进入警戒区、离开警戒区、在警戒区内、穿越围栏、徘徊检测、遗留检测、搬移检测、物品保护、非法停车、快速移动、逆行检测等。

智能事件抓图，即用户订阅智能事件成功后，手动发送一个命令给设备端，设备端抓取当时场景的图片，并通过智能事件上报给用户。

2.14.2 接口总览

表2-14 智能事件上报与抓图接口说明

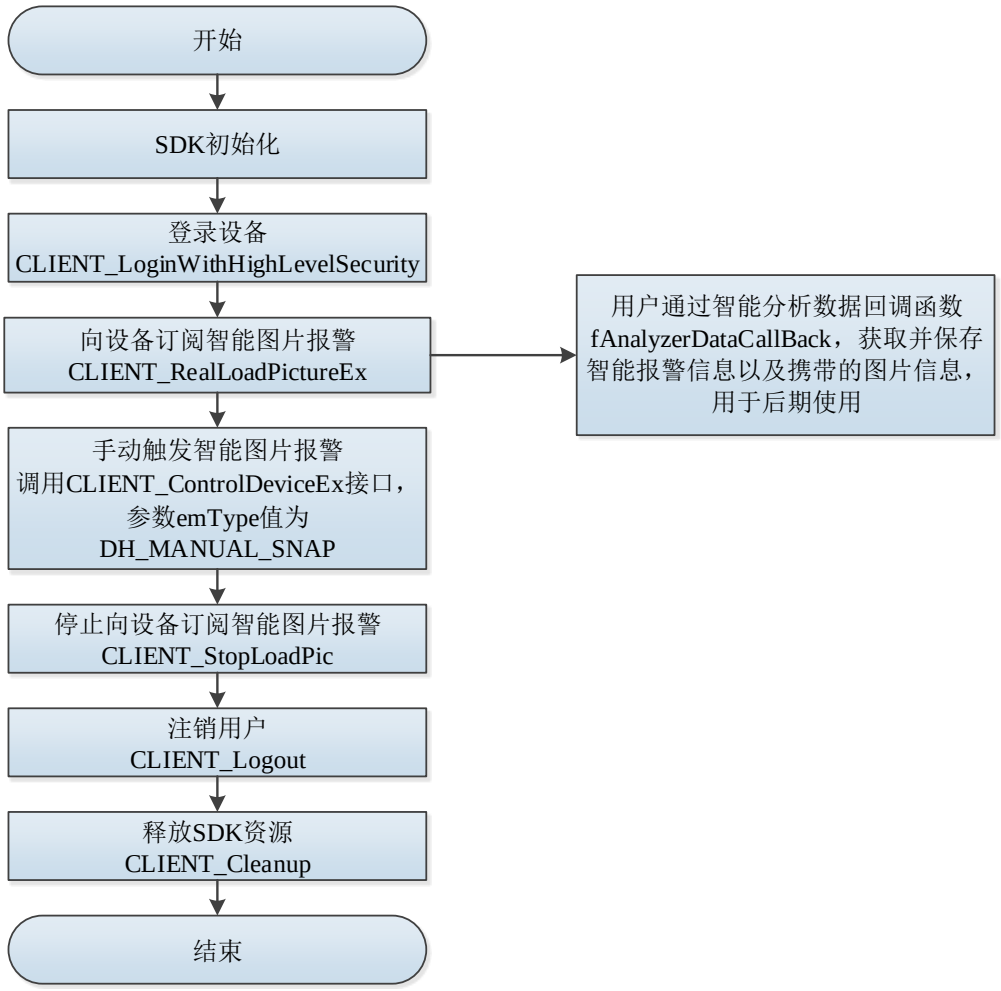
接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_LoginEx2 仍然可以使用，但存在安全风险。所以强烈推荐使用最新接口 CLIENT_LoginWithHighLevelSecurity 登录设备。
CLIENT_RealLoadPictureEx	智能图片报警订阅接口
CLIENT_ControlDeviceEx	设备控制扩展接口
CLIENT_Logout	登出接口

接口	接口说明
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.14.3 流程说明

智能事件上报与抓图流程，如图 2-19 所示。

图2-19 智能事件上报与抓图流程



流程说明

步骤1　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_RealLoadPictureEx，向设备订阅智能抓图报警。订阅成功后，设备上报的智能抓图报警事件通过 fAnalyzerDataCallBack 设置的回调函数通知用户，回调函数的主要任务是显示事件和保存事件。

在回调函数中，用户先根据 SDK 头文件的说明，将入参字符根据事件的类型转化为相应的结构体，再根据自己的需求显示事件或保存事件。

步骤4　若客户有手动触发智能图片报警的需求，则可调用 CLIENT_ControlDeviceEx 接口，参数 emType 值为 DH_MANUAL_SNAP，SDK 会发送命令给设备，设备抓取当前预览图片，通过智能图片报警上报给用户。

步骤5　智能图片报警上报功能使用完毕后，调用 CLIENT_StopLoadPic 停止向设备订阅智能图片报警。

步骤6　业务使用完后，调用 CLIENT_Logout 退出设备。

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

2.14.4 示例代码

```
#include <windows.h>
#include <stdio.h>
#include <list>
#include <time.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lRealLoadHandle = 0L;
static char g_szDevIp[32] = "192.168.4.12";
static int   g_nPort = 37777; // tcp 连接端口，需与期望登录设备页面 tcp 端口配置一致
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//********************************************************************************
// 常用回调集合声明
// 设备断线回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 会调用该函数
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser);

// 断线重连成功回调函数
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_SetAutoReconnect 设置该回调函数，当已断线的设备重连成功时，SDK 会调用
该函数
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser);

// 智能分析数据回调
// 不建议在该回调函数中调用 SDK 接口
// 通过 CLIENT_RealLoadPictureEx/CLIENT_RealLoadPicture 设置该回调函数，当设备端有智能
图片事件上报时，SDK 会调用该函数
// nSequence 表示上传的相同图片情况，为 0 时表示是第一次出现，为 2 表示最后一次出现或仅
出现一次，为 1 表示此次之后还有
// int nState = *(int*) reserved 表示当前回调数据的状态, 为 0 表示当前数据为实时数据，为 1
表示当前回调数据是离线数据，为 2 时表示离线数据传送结束
// 返回值已废除，无特殊意义
int   CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void*
pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void
```

```
*reserved);

//*************************************************************************
void InitTest()
{
    // 初始化 SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // 获取 SDK 版本信息
    // 此操作为可选操作
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作
    // 此操作为可选操作，但建议用户进行设置
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

        // 设置登录超时时间和尝试次数
    // 此操作为可选操作
    int nWaitTime = 5000;    // 登录请求响应超时时间设置为 5s
    int nTryTimes = 3;        // 登录时尝试建立链接 3 次
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // 设置更多网络参数，NET_PARAM 的 nWaittime，nConnectTryNum 成员与 CLIENT_SetConnectTime 接口设置的登录设备超时时间和尝试次数意义相同
    // 此操作为可选操作
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // 登录时尝试建立链接的超时时间
CLIENT_SetNetworkParam(&stuNetParm);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
```

```
strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);
stInparam.nPort = g_nPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);

    while(0 == g_lLoginHandle)
    {
        // 登录设备
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if (0 == g_lLoginHandle)
        {
            // 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
            // 例如：
            // #define NET_NOT_SUPPORTED_EC(23)                // 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // 用户初次登录设备，需要初始化一些数据才能正常实现业务功能，建议登录后等待一小段时间，具体等待时间因设备而异
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
```

```
    if (0 == g_lLoginHandle)
    {
        return;
    }

    // 订阅智能图片报警
LDWORD dwUser = 0;
int nChannel = 0;
    // 每次设置对应一个通道，并且对应一种类型的事件
    // 如果要设置该通道上传所有类型的事件，可以将参数 dwAlarmType 设置为 EVENT_IVS_ALL
    // 如果需要设置一个通道上传两种事件，那么请调用两次 CLIENT_RealLoadPictureEx，并且传入不同的事件类型
    g_lRealLoadHandle = CLIENT_RealLoadPictureEx(g_lLoginHandle, nChannel, EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, dwUser, NULL);
    if (0 == g_lRealLoadHandle)
    {
        printf("CLIENT_RealLoadPictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
        return;
    }

    // 手动抓图触发智能图片报警功能
    while(1)
    {
        char szGetBuf[64] = "";
        printf("manual snap, \'q\': quit, other: yes\n");
        gets(szGetBuf);
        // 输入'q'表示退出手动抓图触发报警，其他表示触发报警
        if (0 == strncmp(szGetBuf, "q", sizeof(szGetBuf) - 1))
        {
            break;
        }

        MANUAL_SNAP_PARAMETER stuSanpParam = {0};
        stuSanpParam.nChannel = 0;
        memcpy(stuSanpParam.bySequence, "just for test", sizeof(stuSanpParam.bySequence) - 1);
        // 手动抓图触发报警功能，该功能只对 ITC 设备有效
        if (FALSE == CLIENT_ControlDeviceEx(g_lLoginHandle, DH_MANUAL_SNAP, &stuSanpParam))
        {
            printf("CLIENT_ControlDeviceEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
            break;
        }
```

```
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();

    // 停止订阅图片报警
    if (0 != g_lRealLoadHandle)
    {
        if (FALSE == CLIENT_StopLoadPic(g_lRealLoadHandle))
        {
            printf("CLIENT_StopLoadPic Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lRealLoadHandle = 0;
        }
    }

    // 退出设备
    if (0 != g_lLoginHandle)
    {
        if (FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lLoginHandle = 0;
        }
    }

    // 清理初始化资源
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}
```

```
int main()
{
	InitTest();
	RunTest();
	EndTest();
	return 0;
}
//**********************************************************************************
// 常用回调集合定义
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD
dwUser)
{
	printf("Call DisConnectFunc\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
	printf("Call HaveReConnect\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
}

int  CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void*
pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void
*reserved)
{
	if (lAnalyzerHandle != g_lRealLoadHandle)
	{
```

```
        return 0;
    }

    int nAlarmChn = 0;
    switch(dwAlarmType)
    {
        case EVENT_IVS_TRAFFIC_OVERLINE:
            {
                printf("EVENT_IVS_TRAFFIC_OVERLINE event\n");
                DEV_EVENT_TRAFFIC_OVERLINE_INFO* pStuInfo =
(DEV_EVENT_TRAFFIC_OVERLINE_INFO*)pAlarmInfo;
                nAlarmChn = pStuInfo->nChannelID;
                printf("nChannelID[%d]\n", pStuInfo->nChannelID);
            }
            break;
        case EVENT_IVS_PARKINGDETECTION:
            {
                printf("EVENT_IVS_PARKINGDETECTION event\n");
                DEV_EVENT_PARKINGDETECTION_INFO* pStuInfo =
(DEV_EVENT_PARKINGDETECTION_INFO*)pAlarmInfo;
                nAlarmChn = pStuInfo->nChannelID;
                printf("nChannelID[%d]\n", pStuInfo->nChannelID);
            }
            break;
        case EVENT_IVS_TRAFFIC_MANUALSNAP:
            {
                printf("EVENT_IVS_TRAFFIC_MANUALSNAP event\n");
                DEV_EVENT_TRAFFIC_MANUALSNAP_INFO* pStuInfo =
(DEV_EVENT_TRAFFIC_MANUALSNAP_INFO*)pAlarmInfo;
                nAlarmChn = pStuInfo->nChannelID;
                // pStuInfo->szManualSnapNo 应该为 "just for test"
                printf("nChannelID[%d]\n", pStuInfo->nChannelID);
            }
            break;
        default:
            printf("other event type[%d]\n", dwAlarmType);
            break;
    }

    if (dwBufSize > 0 && NULL != pBuffer)
    {
        // 预防同一时间收到多张图片，只通过接收时间来保存图片可能会覆盖，于是通过 i 来
标记
```

```
            static int i;
            char szPicturePath[256] = "";
            time_t stuTime;
            time(&stuTime);
            char szTmpTime[128] = "";
            strftime(szTmpTime, sizeof(szTmpTime) - 1, "%y%m%d_%H%M%S",
gmtime(&stuTime));
            _snprintf(szPicturePath, sizeof(szPicturePath)-1, "%d_%s.jpg", ++i, szTmpTime);

            FILE* pFile = fopen(szPicturePath, "wb");
            if (NULL == pFile)
            {
                  return 0;
            }

            int nWrite = 0;
            while(nWrite != dwBufSize)
            {
                  nWrite += fwrite(pBuffer + nWrite, 1, dwBufSize - nWrite, pFile);
            }

            fclose(pFile);
      }

      return 1;
}
```

2.15 APP 管理

2.15.1 简介

APP 管理包括获取设备状态、获取设备应用程序列表、升级含安装（升级准备、升级数据推送、执行升级），以及启动程序、停止程序、删除程序等。

2.15.2 接口总览

表2-15 APP 管理接口说明

接口	接口说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口 说明 CLIENT_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_StartApp	启动程序
CLIENT_StopApp	停止程序
CLIENT_RemoveApp	删除程序
CLIENT_GetNewDevConfig + CFG_CMD_DEV_GENERRAL	获取文件配置
CLIENT_GetInstalledAppInfo	获取安装的应用列表
CLIENT_QueryDevState	查询设备状态
CLIENT_UpgraderInstall	升级相关接口
CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_PREPAREEX	升级准备
CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_APPEND_DATA	升级数据推送
CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_EXECUTE	执行升级
CLIENT_Logout	登出接口
CLIENT_GetLastError	获取接口调用失败时的错误码接口

2.15.3 流程说明

APP 管理流程，如图 2-20 所示。

流程说明

步骤 1　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_StartApp，向设备发送启动 APP 程序。启动成功后需要查询设备信息，CLIENT_GetNewDevConfig + CFG_CMD_DEV_GENERRAL 接口获取文件配置，CLIENT_QueryDevState 接口查询设备状态。

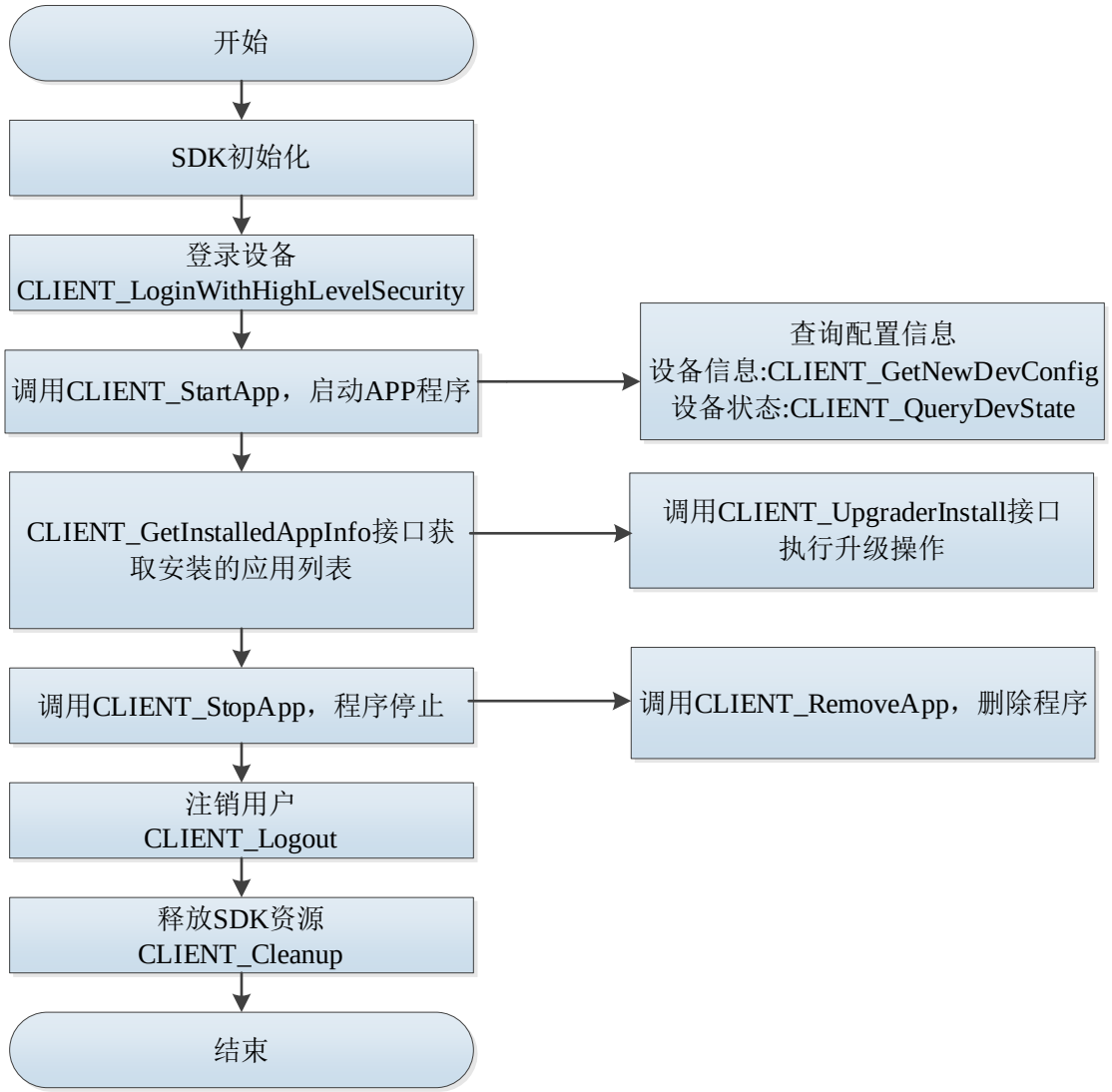
步骤 4　若客户需要升级 APP，则可调用 CLIENT_GetInstalledAppInfo 接口获取安装的应用列表，CLIENT_UpgraderInstall 接口加枚举 emType 用于升级：

参数 emType 值为 EM_UPGRADER_INSTALL_PREPAREEX，SDK 会发送升级准备命令给设备，设备返回成功表示准备升级。

参数 emType 值为 EM_UPGRADER_INSTALL_APPEND_DATA，SDK 会推送升级数据给设备，设备受到升级数据等待升级。
参数 emType 值为 EM_UPGRADER_INSTALL_EXECUTE，SDK 会发送执行升级命令给设备，设备返回成功表示执行升级。
参数 emType 值为 EM_UPGRADER_INSTALL_GETSTATE，SDK 会发送获取升级状态命令给设备，设备返回升级状态。
参数 emType 值为 EM_UPGRADER_INSTALL_CANCEL，SDK 会发送取消升级命令给设备，设备返回成功表示取消升级。
参数 emType 值为 EM_UPGRADER_INSTALL_FIRMWAREEX，SDK 会发送指定升级包地址进行升级命令给设备，设备收到并调用指定升级包进行升级。

步骤 5 调用接口 CLIENT_StopApp，程序停止，若客户需要删除 APP 程序，需调用 CLIENT_RemoveApp 接口。
步骤 6 业务使用完后，调用 CLIENT_Logout 退出设备。
步骤 7 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

图2-20 APP 管理接口流程说明



2.15.4 示例代码

```
#include "stdafx.h"
#include <iostream>
#include <cstring>
#include "dhnetsdk.h"
#include "dhconfigsdk.h"
#include "testUtil.h"

#ifdef _WIN64
#pragma comment(lib, "./lib/win64/dhnetsdk.lib")
#pragma comment(lib, "./lib/win64/dhconfigsdk.lib")
#else
#pragma comment(lib, "./lib/win32/dhnetsdk.lib")
#pragma comment(lib, "./lib/win32/dhconfigsdk.lib")
#endif
LLONG g_lHandle = 0;
LLONG g_lFindID = 0;
LLONG lRealloadHandle = 0;
LLONG g_lFindHandle = 0;

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
	printf("Device is DisConnected\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD
dwUser)
{
	printf("Device is Reconnecting!\n");
}
void TestStartApp(void* data)
{
	LLONG lLoginID = *(LLONG*)data;
	NET_IN_START_APP stuIn = {sizeof(NET_IN_START_APP)};;
	memset(&stuIn, 0, sizeof(NET_IN_START_APP));
	stuIn.dwSize = sizeof(NET_IN_START_APP);
	NET_OUT_START_APP stuOut = {sizeof(NET_OUT_START_APP)};;
	memset(&stuOut, 0, sizeof(NET_OUT_START_APP));
	stuOut.dwSize = sizeof(NET_OUT_START_APP);
	stuIn.nAppID = 3214;
	strncpy(stuIn.szAppName, "TestAppCheck", sizeof(stuIn.szAppName) - 1);
```

```
    BOOL bRet = CLIENT_StartApp(lLoginID, &stuIn,&stuOut,5000);
    if(bRet)
    {
        printf("CLIENT_StartApp success\r\n");
    }
    else
    {
        printf("CLIENT_StartApp fail, error:%0x\n", CLIENT_GetLastError());
    }
}
void TestQueryDevState(void* data)
{
    LLONG lLoginID = *(LLONG*)data;
    DHDEV_UPGRADE_STATE_INFO *pstInfo = new DHDEV_UPGRADE_STATE_INFO;
    int nRetLen = sizeof(DHDEV_UPGRADE_STATE_INFO);
    BOOL bRet = CLIENT_QueryDevState(lLoginID, DH_DEVSTATE_GET_UPGRADE_STATE, (char
*)pstInfo, sizeof(DHDEV_UPGRADE_STATE_INFO), &nRetLen, 1000);
    if(bRet)
    {
        printf("CLIENT_QueryDevState success\r\n");
    }
    else
    {
        printf("CLIENT_QueryDevState fail, error:%0x\n", CLIENT_GetLastError());
    }
    delete pstInfo;
    pstInfo = NULL;
}
void TestGetNewDevConfig(void* data)
{
    LLONG lLoginID = *(LLONG*)data;
    int nChannel = -1;
    int nBufLen = 1024*1024*20;
    char *pbufsize = new char[nBufLen];
    memset(pbufsize, 0, nBufLen);
    int nError = 0;
    int nRetlen = 0;
    CFG_DEV_DISPOSITION_INFO *pstinfo = new CFG_DEV_DISPOSITION_INFO;
    memset(pstinfo, 0, sizeof(CFG_DEV_DISPOSITION_INFO));

    BOOL bRet = CLIENT_GetNewDevConfig(lLoginID, CFG_CMD_DEV_GENERRAL, nChannel,
pbufsize, nBufLen, &nError, 5000);
```

```
    if (bRet)
    {
        printf("GetNewDevConfig success\n");
        bRet = CLIENT_ParseData(CFG_CMD_DEV_GENERRAL, pbufsize, pstinfo,
sizeof(CFG_DEV_DISPOSITION_INFO), NULL);
        if (bRet)
        {
            printf("ParseData success\n");
            printf("pstinfo->nLocalNo :%d\n",pstinfo->nLocalNo);
            printf("pstinfo->szMachineID :%s\n",pstinfo->szMachineID);

        }
        else
        {
            printf("ParseData Failed, error is %d\n", CLIENT_GetLastError()&0x7fffffff);
        }
    }
    else
    {
        printf("GetNewDevConfig Failed, error is %d\n", CLIENT_GetLastError()&0x7fffffff);
    }
    delete[] pbufsize;
    pbufsize = NULL;
    delete pstinfo;
    pstinfo = NULL;
}
void TestGetInstalledAppInfo(void* data)
{
    LLONG lLoginID = *(LLONG*)data;
    NET_IN_GET_INSTALLED_APP_INFO stuIn = {sizeof(NET_IN_GET_INSTALLED_APP_INFO)};;
    memset(&stuIn, 0, sizeof(NET_IN_GET_INSTALLED_APP_INFO));
    stuIn.dwSize = sizeof(NET_IN_GET_INSTALLED_APP_INFO);
    NET_OUT_GET_INSTALLED_APP_INFO stuOut =
{sizeof(NET_OUT_GET_INSTALLED_APP_INFO)};;
    memset(&stuOut, 0, sizeof(NET_OUT_GET_INSTALLED_APP_INFO));
    stuOut.dwSize = sizeof(NET_OUT_GET_INSTALLED_APP_INFO);

    BOOL bRet = CLIENT_GetInstalledAppInfo(lLoginID, &stuIn,&stuOut,5000);
    if(bRet)
    {
        printf("CLIENT_GetInstalledAppInfo success\r\n");
        printf("stuOut.nListCount :%d\n",stuOut.nListCount);
        for(int i = 0;i<stuOut.nListCount;i++)
```

```
            {

        printf("stuOut.stuAppInfoList[%d].szAppName :%s\n",i,stuOut.stuAppInfoList[i].szAppName
);
            }
        }
        else
        {
            printf("CLIENT_GetInstalledAppInfo fail, error:%0x\n", CLIENT_GetLastError());
        }
}
void TestUpgraderInstallPrepare(void* data)
{
        LLONG lLoginID = *(LLONG*)data;
        EM_NET_UPGRADE_INSTALL_TYPE emType = EM_UPGRADER_INSTALL_PREPAREEX;

        NET_IN_INSTALL_PREPAREEX stuIn = {sizeof(NET_IN_INSTALL_PREPAREEX)};
        memset(&stuIn, 0, sizeof(NET_IN_INSTALL_PREPAREEX));
        stuIn.dwSize = sizeof(NET_IN_INSTALL_PREPAREEX);
        NET_OUT_INSTALL_PREPAREEX stuOut = {sizeof(NET_OUT_INSTALL_PREPAREEX)};
        memset(&stuOut, 0, sizeof(NET_OUT_INSTALL_PREPAREEX));
        stuOut.dwSize = sizeof(NET_OUT_INSTALL_PREPAREEX);
        stuIn.bReliable = true;
        stuIn.emNextOperate = EM_NET_NEXT_OPERATE_INSTALL;
        stuIn.nTotalLength = 1024;
        strncpy(stuIn.szAppName, "TestAppCheck", sizeof(stuIn.szAppName) - 1);

        BOOL bRet = CLIENT_UpgraderInstall(lLoginID, emType,&stuIn,&stuOut,5000);
        if(bRet)
        {
            printf("TestUpgraderInstallPrepare success\r\n");
        }
        else
        {
            printf("TestUpgraderInstallPrepare fail, error:%0x\n", CLIENT_GetLastError());
        }
}
void TestUpgraderInstallAppend(void* data)
{
        LLONG lLoginID = *(LLONG*)data;
        EM_NET_UPGRADE_INSTALL_TYPE emType = EM_UPGRADER_INSTALL_APPEND_DATA;
        NET_IN_INSTALL_APPEND_DATA stuIn = {sizeof(NET_IN_INSTALL_APPEND_DATA)};
        memset(&stuIn, 0, sizeof(NET_IN_INSTALL_APPEND_DATA));
```

```
    stuIn.dwSize = sizeof(NET_IN_INSTALL_APPEND_DATA);
    NET_OUT_INSTALL_APPEND_DATA stuOut = {sizeof(NET_OUT_INSTALL_APPEND_DATA)};
    memset(&stuOut, 0, sizeof(NET_OUT_INSTALL_APPEND_DATA));
    stuOut.dwSize = sizeof(NET_OUT_INSTALL_APPEND_DATA);
    stuIn.nTotalLen = 1024;
    stuIn.nAppendLen = 32;
    stuIn.pAppendData = new unsigned char[stuIn.nAppendLen];
    for(int i = 0;i<stuIn.nAppendLen;i++)
    {
        stuIn.pAppendData[i] = (unsigned char)(i+1);
    }

    BOOL bRet = CLIENT_UpgraderInstall(lLoginID, emType,&stuIn,&stuOut,5000);
    if(bRet)
    {
        printf("TestUpgraderInstallAppend success\r\n");
    }
    else
    {
        printf("TestUpgraderInstallAppend fail, error:%0x\n", CLIENT_GetLastError());
    }
    delete[] stuIn.pAppendData;
    stuIn.pAppendData = NULL;
}
void TestUpgraderInstallExecute(void* data)
{
    LLONG lLoginID = *(LLONG*)data;
    EM_NET_UPGRADE_INSTALL_TYPE emType = EM_UPGRADER_INSTALL_EXECUTE;

    NET_IN_INSTALL_EXECUTE stuIn = {sizeof(NET_IN_INSTALL_EXECUTE)};
    memset(&stuIn, 0, sizeof(NET_IN_INSTALL_EXECUTE));
    stuIn.dwSize = sizeof(NET_IN_INSTALL_EXECUTE);
    NET_OUT_INSTALL_EXECUTE stuOut = {sizeof(NET_OUT_INSTALL_EXECUTE)};
    memset(&stuOut, 0, sizeof(NET_OUT_INSTALL_EXECUTE));
    stuOut.dwSize = sizeof(NET_OUT_INSTALL_EXECUTE);
    stuIn.bAutoReboot = false;

    BOOL bRet = CLIENT_UpgraderInstall(lLoginID, emType,&stuIn,&stuOut,5000);
    if(bRet)
    {
        printf("TestUpgraderInstallExecute success\r\n");
    }
    else
```

```
    {
        printf("TestUpgraderInstallExecute fail, error:%0x\n", CLIENT_GetLastError());
    }
}
void TestStopApp(void* data)
{
    LLONG lLoginID = *(LLONG*)data;
    NET_IN_STOP_APP stuIn = {sizeof(NET_IN_STOP_APP)};;
    memset(&stuIn, 0, sizeof(NET_IN_STOP_APP));
    stuIn.dwSize = sizeof(NET_IN_STOP_APP);
    NET_OUT_STOP_APP stuOut = {sizeof(NET_OUT_STOP_APP)};;
    memset(&stuOut, 0, sizeof(NET_OUT_STOP_APP));
    stuOut.dwSize = sizeof(NET_OUT_STOP_APP);
    stuIn.nAppID = 3214;
    strncpy(stuIn.szAppName, "TestAppCheck", sizeof(stuIn.szAppName) - 1);

    BOOL bRet = CLIENT_StopApp(lLoginID, &stuIn,&stuOut,5000);
    if(bRet)
    {
        printf("CLIENT_StopApp success\r\n");
    }
    else
    {
        printf("CLIENT_StopApp fail, error:%0x\n", CLIENT_GetLastError());
    }
}

void TestRemoveApp(void* data)
{
    LLONG lLoginID = *(LLONG*)data;
    NET_IN_REMOVE_APP stuIn = {sizeof(NET_IN_REMOVE_APP)};;
    memset(&stuIn, 0, sizeof(NET_IN_REMOVE_APP));
    stuIn.dwSize = sizeof(NET_IN_REMOVE_APP);
    NET_OUT_REMOVE_APP stuOut = {sizeof(NET_OUT_REMOVE_APP)};;
    memset(&stuOut, 0, sizeof(NET_OUT_REMOVE_APP));
    stuOut.dwSize = sizeof(NET_OUT_REMOVE_APP);
    stuIn.nAppID = 3214;
    strncpy(stuIn.szAppName, "TestAppCheck", sizeof(stuIn.szAppName) - 1);

    BOOL bRet = CLIENT_RemoveApp(lLoginID, &stuIn,&stuOut,5000);
    if(bRet)
    {
        printf("CLIENT_RemoveApp success\r\n");
```

```
    }
    else
    {
        printf("CLIENT_RemoveApp fail, error:%0x\n", CLIENT_GetLastError());
    }
}

int main()
{
    SetConsoleOutputCP(CP_UTF8);
    CLIENT_Init(DisConnectFunc, 0);
    CLIENT_SetAutoReconnect(HaveReConnect, NULL);

    LOG_SET_PRINT_INFO    stLogPrintInfo = {sizeof(stLogPrintInfo)};
    CLIENT_LogOpen(&stLogPrintInfo);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
    memset(&stInparam, 0, sizeof(stInparam));
    stInparam.dwSize = sizeof(stInparam);
    strncpy(stInparam.szIP, "10.33.121.107", sizeof(stInparam.szIP) - 1);
    strncpy(stInparam.szPassword, "admin123", sizeof(stInparam.szPassword) - 1);
    strncpy(stInparam.szUserName, "admin", sizeof(stInparam.szUserName) - 1);
    stInparam.nPort = 37777;
    stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    printf("stInparam.szIP: %s, stInparam.nPort: %d\n", stInparam.szIP,stInparam.nPort);

    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
    memset(&stOutparam, 0, sizeof(stOutparam));
    stOutparam.dwSize = sizeof(stOutparam);
    LLONG lLoginID = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);
    if (0 == lLoginID)
    {
        printf("CLIENT_LoginWithHighLevelSecurity fail,error:%d\n", stOutparam.nError);
        printf("Enter any key to exit\n");
        getchar();
        CLIENT_Cleanup();
        return 0;
    }
    else
    {
        CTestMenu menu(&lLoginID);
        menu.addItem("case: CLIENT_StartApp.", TestStartApp);
```

```
            menu.addItem("case: CLIENT_QueryDevState.", TestQueryDevState);
            menu.addItem("case: CLIENT_GetNewDevConfig + CFG_CMD_DEV_GENERRAL.",
TestGetNewDevConfig);
            menu.addItem("case: CLIENT_GetInstalledAppInfo.", TestGetInstalledAppInfo);
           menu.addItem("case: CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_PREPAREEX.",
TestUpgraderInstallPrepare);
            menu.addItem("case: CLIENT_UpgraderInstall +
EM_UPGRADER_INSTALL_APPEND_DATA.", TestUpgraderInstallAppend);
            menu.addItem("case: CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_EXECUTE.",
TestUpgraderInstallExecute);
            menu.addItem("case: CLIENT_StopApp.", TestStopApp);
            menu.addItem("case: CLIENT_RemoveApp.", TestRemoveApp);
            menu.run();
            CLIENT_Logout(lLoginID);
        }
        CLIENT_Cleanup();

        return 0;
}
```

第 3 章 回调函数定义

3.1 断线回调函数 fDisConnect

表3-1 fDisConnect

选项	说明
接口描述	断线回调函数，当已登录的设备出现断线时，通过该接口通知用户。
前置条件	无
函数	typedef void(CALLBACK *fDisConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);
参数	● lLoginID 设备登录 ID，对应 CLIENT_LoginWithHighLevelSecurity 接口的返回值。 ● pchDVRIP 设备 IP，对应断线的设备 IP，与登录时入参中的设备 IP 一致。 ● nDVRPort 端口，对应断线的设备端口，与登录时入参中的设备端口一致。 ● dwUser 用户数据，与用户在设置 fDisConnect 回调函数时传入的用户数据一致。
返回值	无
注释	在 CLIENT_Init 接口中设置该回调函数 用户可通过参数（lLoginID、pchDVRIP 和 nDVRPort）定位到是哪次登录的设备出现了断线。

3.2 断线重连成功回调函数 fHaveReConnect

表3-2 fHaveReConnect

选项	说明
接口描述	断线重连成功回调函数，当已断线设备重连成功时，通过该接口通知用户。
前置条件	无
函数	typedef void (CALLBACK *fHaveReConnect) (LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

选项	说明
参数	● lLoginID 设备登录 ID。对应 CLIENT_LoginWithHighLevelSecurity 接口的返回值。 ● pchDVRIP 设备 IP，对应断线的设备 IP，与登录时入参中的设备 IP 一致。 ● nDVRPort 端口，对应断线的设备端口，与登录时入参中的设备端口一致。 ● dwUser 用户数据，与用户在设置 fHaveReConnect 回调函数时传入的用户数据一致。
返回值	无
注释	在 CLIENT_SetAutoReconnect 接口中设置该回调函数。 用户可通过 lLoginID、pchDVRIP 和 nDVRPort 三个参数，可以定位到是哪次断线的设备重新连接成功了。

3.3 实时预览数据回调函数 fRealDataCallBackEx

表3-3 fRealDataCallBackEx

选项	说明
接口描述	实时预览数据回调函数原型扩展
前置条件	无
函数	typedef void (CALLBACK *fRealDataCallBackEx)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser);

选项	说明
参数	● lRealHandle 实时预览句柄。CLIENT_RealPlayEx 等拉取实时预览码流接口的返回值。 ● dwDataType 标识回调出来的数据类型。 由 CLIENT_SetRealDataCallBackEx 接口的 dwFlag 决定。dwDataType 具体的数据类型定义如下： ◇ 0：原始数据（与 SaveRealData 保存的数据一致） ◇ 1：帧数据 ◇ 2：yuv 数据 ◇ 3：pcm 音频数据 ● pBuffer 回调数据。根据数据类型的不同每次回调不同长度的数据，除了原始数据，其他数据类型都是按帧格式回调，每次回调一帧数据。 ● dwBufSize 回调数据的长度。不同的数据类型，长度不同（单位：字节）。 ● param 回调数据参数结构体。 根据不同的数据类型，参数结构也不一致。 ◇ 当类型为 0（原始数据）和 2（YUV 数据）时，param 为 0。 ◇ 当回调的数据类型为帧数据时，param 为一个 tagVideoFrameParam 结构体指针，具体请参见“4.8 视频数据帧信息结构体 tagVideoFrameParam”。 ◇ 当数据类型是 PCM 数据时，param 为一个 tagCBPCMDataParam 结构体指针，具体请参见“4.9 音频数据帧信息结构体 tagCBPCMDataParam”。 ● dwUserData 用户数据，与用户在设置 fRealDataCallBackEx 回调函数时传入的用户数据一致。
返回值	无
注释	在 CLIENT_SetRealDataCallBackEx 接口中设置该回调函数。 用户在该回调函数中可通过参数 lRealHandle 来唯一识别是哪次实时预览的回调数据。

3.4 实时预览数据回调函数 fRealDataCallBackEx2

表3-4 fRealDataCallBackEx2

选项	说明
接口描述	实时预览数据回调函数原型扩展 2
前置条件	无
函数	typedef void (CALLBACK *fRealDataCallBackEx2)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser);

选项	说明
参数	● lRealHandle 实时预览句柄。CLIENT_RealPlayEx 等拉取实时预览码流接口的返回值。 ● dwDataType 标识回调出来的数据类型。 由 CLIENT_SetRealDataCallBackEx 接口的 dwFlag 决定。dwDataType 具体的数据类型定义如下： ◇ 0：原始数据（与 SaveRealData 保存的数据一致） ◇ 1：帧数据 ◇ 2：yuv 数据 ◇ 3：pcm 音频数据 ● pBuffer 回调数据。根据数据类型的不同每次回调不同长度的数据，除了原始数据，其他数据类型都是按帧格式回调，每次回调一帧数据。 ● dwBufSize 回调数据的长度。不同的数据类型，长度不同（单位：字节）。 ● param 回调数据参数结构体。 根据不同的数据类型，参数结构也不一致。 ◇ 当类型为 0（原始数据）和 2（YUV 数据）时，param 为 0。 ◇ 当回调的数据类型为帧数据时，param 为一个 tagVideoFrameParam 结构体指针，具体请参见“4.8 视频数据帧信息结构体 tagVideoFrameParam”。 ◇ 当数据类型是 PCM 数据时，param 为一个 tagCBPCMDataParam 结构体指针，具体请参见“4.9 音频数据帧信息结构体 tagCBPCMDataParam”。 ● dwUserData 用户数据，与用户在设置 fRealDataCallBackEx 回调函数时传入的用户数据一致。
返回值	无
注释	无

3.5 实时预览转码数据回调函数 fDataCallBackEx

表3-5 fDataCallBackEx

选项	说明
接口描述	实时预览转码数据回调函数原型扩展 2
前置条件	无
函数	typedef void (CALLBACK *fDataCallBackEx)(LLONG lRealHandle, NET_DATA_CALL_BACK_INFO *pDataCallBack, LDWORD dwUser);

选项	说明
参数	● lRealHandle 实时预览句柄。CLIENT_RealPlayEx 等拉取实时预览码流接口的返回值。 ● pDataCallBack 回调的数据。见结构体 NET_DATA_CALL_BACK_INFO ● dwUserData 用户数据，与用户在设置 fRealDataCallBackEx 回调函数时传入的用户数据一致。
返回值	无
注释	无

3.6 回放进度回调函数 fDownLoadPosCallBack

表3-6 fDownLoadPosCallBack

选项	说明
接口描述	回放进度回调函数
前置条件	无
函数	typedef void (CALLBACK *fDownLoadPosCallBack)(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);
参数	● lPlayHandle 录像回放句柄。CLIENT_PlayBackByTimeEx 等录像回放接口的返回值。 ● dwTotalSize 指本次播放总大小，单位为 KB。 ● dwDownLoadSize 指已经播放的大小，单位为 KB。 ◇ -1：表示本次回放结束。 ◇ -2：表示写文件失败。 ● dwUser 用户数据，与用户在设置 fDownLoadPosCallBack 回调函数时传入的用户数据一致。
返回值	无
注释	在 CLIENT_PlayBackByTimeEx 等录像回放接口中设置该回调函数。 用户在该回调函数中可通过参数 lPlayHandle 来唯一识别是哪次拉流对应的进度回调。

3.7 录像回放数据回调函数 fDataCallBack

表3-7 fDataCallBack

选项	说明
接口描述	录像回放数据回调函数
前置条件	无

选项	说明
函数	typedef int (CALLBACK *fDataCallBack)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);
参数	● lRealHandle 录像回放句柄。CLIENT_PlayBackByTimeEx 等录像回放接口的返回值。 ● dwDataType 数据类型，该参数在此回调中始终为 0，表示录像为原始数据。 ● pBuffer 数据缓冲，用于存放本次回调的录像数据。 ● dwBufSize 缓冲长度（单位字节）。 ● dwUser 用户数据，与用户在设置 fDataCallBack 回调函数时传入的用户数据一致。
返回值	0：表示本次回调失败，下次回调会返回相同的数据。 1：表示本次回调成功，下次回调会返回后续的数据。
注释	在 CLIENT_PlayBackByTimeEx 等录像回放接口中设置该回调函数。 设置该回调函数时，若对应的 hWnd 参数不为 NULL，则不管回调函数返回值为多少都认为回调成功，下次回调会返回后续的数据。 用户在该回调函数中可通过参数 lRealHandle 来唯一识别是哪次拉流对应的回调数据。

3.8 录像下载进度回调函数 fTimeDownLoadPosCallBack

表3-8 fTimeDownLoadPosCallBack

选项	说明
接口描述	按时间录像下载进度回调函数
前置条件	无
函数	typedef void (CALLBACK *fTimeDownLoadPosCallBack) (LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser);

选项	说明
参数	● lPlayHandle 录像下载句柄。CLIENT_DownloadByTimeEx 等录像回放接口的返回值。 ● dwTotalSize 本次下载总大小，单位为 KB。 ● dwDownLoadSize 已下载大小，单位为 KB。 ● index 当前下载的录像文件序号，从 0 开始。 ● recordfileinfo 当前下载录像文件信息。 具体请参见 NET_RECORDFILE_INFO 结构体说明。 ● dwUser 用户数据，与用户在设置 fTimeDownLoadPosCallBack 回调函数时传入的用户数据一致。
返回值	无
注释	在 CLIENT_DownloadByTimeEx 等按时间下载录像接口中设置该回调函数。 用户在该回调函数中可通过参数 lPlayHandle 来唯一识别是哪次录像下载对应的进度回调。

3.9 报警上报回调 fMessCallBack

表3-9 fMessCallBack

选项	说明
接口描述	报警上报回调函数原型
前置条件	无
函数	typedef BOOL (CALLBACK *fMessCallBack)(LONG lCommand, LLONG lLoginID, char *pBuf, DWORD dwBufLen, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

选项	说明
参数	● lCommand 回调的报警事件类型。与 pBuf 匹配使用，lCommand 值不同，则 pBuf 指向的数据类型不同。具体请参见该表格注释项的描述。 ● lLoginID 设备登录 ID。对应 CLIENT_LoginWithHighLevelSecurity 等设备登录接口的返回值。 ● pBuf 接收报警数据的缓存。根据调用的侦听接口和 lCommand 值不同，pBuf 指向的数据类型不同。具体请参见该表格注释项的描述。 ● dwBufLen 接收报警数据缓存的长度（单位：字节）。 ● pchDVRIP 报警上报的设备 IP。 ● nDVRPort 报警上报的设备端口。 ● dwUser 用户数据，与用户在设置 fMessCallBack 回调函数时传入的用户数据一致。
返回值	成功返回 TRUE，失败返回 FALSE
注释	所有登录的设备统一使用一个报警上报回调函数。 用户通过参数：lLoginID 来定位到是哪次登录对应的报警上报。 lCommand 值不同，pBuf 指向的数据类型不同 由于报警事件类型过多，此处不一一举例，用户可在 dhnetsdk.h 中搜索如下字段： // 扩展报警类型,对应 CLIENT_StartListenEx 接口 #define DH_ALARM_ALARM_EX　0x2101　// 外部报警 即可找到相应的说明

3.10 搜索设备回调函数 fSearchDevicesCB

表3-10 fSearchDevicesCB

选项	说明
接口描述	搜索设备回调原型
前置条件	无
函数	typedef void (CALLBACK *fSearchDevicesCB)(DEVICE_NET_INFO_EX *pDevNetInfo, void* pUserData);
参数	● pDevNetInfo 设备信息结构体。具体请参见 DEVICE_NET_INFO_EX 结构体定义说明。 ● pUserData 用户数据。与用户在设置 fSearchDevicesCB 回调函数时传入的用户数据一致。
返回值	无
注释	搜索设备回调。 不建议在该回调函数中调用 SDK 接口。 通过 CLIENT_StartSearchDevices/CLIENT_SearchDevicesByIPs 设置该回调函数，当搜索到设备时，SDK 会调用该函数。

3.11 搜索设备扩展回调函数 fSearchDevicesCBEx

表3-11 fSearchDevicesCBEx

选项	说明
接口描述	搜索设备回调原型
前置条件	无
函数	typedef void(CALLBACK * fSearchDevicesCBEx)(LLONG lSearchHandle, DEVICE_NET_INFO_EX2 *pDevNetInfo, void* pUserData);
参数	● lSearchHandle 搜索句柄。 ● pDevNetInfo 设备信息结构体。具体请参见 DEVICE_NET_INFO_EX 结构体定义说明。 ● pUserData 用户数据。与用户在设置 fSearchDevicesCB 回调函数时传入的用户数据一致。
返回值	无
注释	搜索设备回调。 不建议在该回调函数中调用 SDK 接口。 通过 CLIENT_StartSearchDevicesEx/CLIENT_SearchDevicesByIPs 设置该回调函数，当搜索到设备时，SDK 会调用该函数。

3.12 智能图片报警回调函数 fAnalyzerDataCallBack

表3-12 fAnalyzerDataCallBack

选项	说明
接口描述	智能图片报警回调原型
前置条件	无
函数	typedef int (CALLBACK *fAnalyzerDataCallBack)(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved);

选项	说明
参数	● lAnalyzerHandle 智能图片报警订阅句柄。当多个智能图片报警订阅使用相同的回调函数时，可以通过 lAnalyzerHandle 唯一定位到相应的订阅操作。 ● dwAlarmType 智能图片报警类型。与 pAlarmInfo 匹配使用，dwAlarmType 值不同，则 pAlarmInfo 指向的数据类型不同。具体请参见该表格注释项的描述。 ● pAlarmInfo 结构体指针。与 dwAlarmType 匹配使用，dwAlarmType 值不同，则 pAlarmInfo 指向的数据类型不同。具体请参见该表格注释项的描述。 ● pBuffer 智能图片信息缓存。 ● dwBufSize 智能图片信息大小。 ● dwUser 用户数据。与用户在设置 fSearchDevicesCB 回调函数时传入的用户数据一致。 ● nSequence 图片是否重复识别字段： ◇ 0：表示该图片是第一次出现，且后续有报警使用相同的图片。 ◇ 1：表示该图片与前一个报警中的图片一样，且后续还有报警使用相同的图片。 ◇ 2：表示该图片与前一个报警中的图片一样，但是是最后一次出现或者表示该图片仅出现一次（大部分报警都是有单独对应的图片，nSequence 值一般都为 2）。 ● reserved 当前回调数据的状态，reserved 为 int 型指针。 *(int *)reserved 值说明： ◇ 0：表示当前数据为实时数据。 ◇ 1：表示当前回调数据是离线数据。 ◇ 2：表示离线数据传送结束（大部分智能图片报警的数据都为实时数据， *(int *)reserved 一般为 0）。
返回值	返回值已废除，无特殊意义，用户返回 0 即可。
注释	智能图片报警回调 不建议在该回调函数中调用 SDK 接口 通过 CLIENT_RealLoadPictureEx/CLIENT_RealLoadPicture 设置该回调函数，当设备端有智能图片事件上报时，SDK 会调用该函数 dwAlarmType 值不同，pAlarmInfo 指向的数据类型不同 由于智能报警事件类型过多且一直在不断增加，这里就不一一举例，用户可在 dhnetsdk.h 中搜索如下字段： // 智能分析事件类型 #define EVENT_IVS_ALL 0x00000001 // 订阅所有事件 即可找到相应的说明

3.13 视频抓图回调函数 fSnapRev

表3-13 fSnapRev

选项	说明
接口描述	前端视频抓图回调函数原型
前置条件	无

选项	说明
函数	typedef void (CALLBACK *fSnapRev)(LLONG lLoginID, BYTE *pBuf, UINT RevLen, UINT EncodeType, DWORD CmdSerial, LDWORD dwUser);
参数	● lLoginID 设备登录句柄。当多个前端视频抓图使用相同的回调函数时，可通过 lLoginID 唯一定位到相应的抓图操作。 ● pBuf 图片信息缓存。用于存储设备返回的前端抓图的图片数据。pBuf 内存由 SDK 内部申请释放。 ● RevLen 图片信息缓存大小。 ● EncodeType 编码类型。 ◇ 10：表示 jpeg 图片 ◇ 0：mpeg4 的 I 帧。 ● CmdSerial 抓图序列号。由调用抓图请求接口 CLIENT_SnapPictureEx 时通过入参传入。 ● dwUser 用户数据。与用户在设置 fSnapRev 回调函数时传入的用户数据一致。
返回值	无
注释	抓图回调函数 不建议在该回调函数中调用 SDK 接口 通过 CLIENT_SetSnapRevCallBack 设置该回调函数，当前端设备有抓图数据发送过来时，SDK 会调用该函数

3.14 视频预览断开回调函数 fRealPlayDisConnect

表3-14 fRealPlayDisConnect

选项	说明
接口描述	视频预览断开回调函数原型
前置条件	无
函数	typedef void (CALLBACK *fRealPlayDisConnect)(LLONG lOperateHandle, EM_REALPLAY_DISCONNECT_EVENT_TYPE dwEventType, void* param, LDWORD dwUser);

选项	说明
参数	● lOperateHandle 实时预览句柄。当多个实时预览设备使用相同的回调函数时，可通过 lOperateHandle 唯一定位到相应的操作。 ● dwEventType 链接失败原因。具体内容请参见 EM_REALPLAY_DISCONNECT_EVENT_TYPE 枚举说明。 ● param 保留字段，默认为 NULL。 ● dwUser 用户数据。与用户在设置 fRealPlayDisConnect 回调函数时传入的用户数据一致。
返回值	无
注释	视频预览断开回调函数 不建议在该回调函数中调用 SDK 接口 通过 CLIENT_RealPlayEx 设置该回调函数，实时预览链接断开时，SDK 会调用该函数

3.15 音频数据回调函数 pfAudioDataCallBack

表3-15 pfAudioDataCallBack

选项	说明
接口描述	语音对讲的音频数据回调函数原形
前置条件	无
函数	typedef void (CALLBACK *pfAudioDataCallBack)(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag, LDWORD dwUser);
参数	● lTalkHandle 语音对讲句柄。CLIENT_StartTalkEx 等打开语音对讲接口的返回值。 ● pDataBuf 回调出来的音频数据。具体数据来源由参数 byAudioFlag 决定。 ● dwBufSize 回调出来的音频数据的长度(单位:字节)。 ● byAudioFlag 音频数据归属标志 ◇ 0：表示收到本地录音库采集的 PC 端音频数据，调用过 CLIENT_RecordStartEx 接口后，才会收到该类型数据的回调。 ◇ 1：表示收到的设备端发过来的音频数据。 ● dwUser 用户数据，与用户在设置 pfAudioDataCallBack 回调函数时传入的用户数据一致。
返回值	无
注释	在 CLIENT_StartTalkEx 等打开语音对讲接口中设置该回调函数

第 4 章 结构体定义

4.1 设备信息结构体 NET_DEVICEINFO

表4-1 NET_DEVICEINFO

选项	说明
结构体描述	设备信息结构体

结构体	typedef struct
	{
	BYTE sSerialNumber[DH_SERIALNO_LEN];
	BYTE byAlarmInPortNum;
	BYTE byAlarmOutPortNum;
	BYTE byDiskNum;
	BYTE byDVRType;
	union
	{
	BYTE byChanNum;
	BYTE byLeftLogTimes;
	};
	} NET_DEVICEINFO, *LPNET_DEVICEINFO;

成员	● sSerialNumber 序列号。 ● byAlarmInPortNum DVR 报警输入个数。 ● byAlarmOutPortNum DVR 报警输出个数。 ● byDiskNum DVR 硬盘个数。 ● byDVRType DVR 类型，请参见“4.1NET_DEVICE_TYPE”。 ● byChanNum DVR 通道个数，登录成功时有效。 ● byLeftLogTimes 当登录失败原因为密码错误时，通过此参数通知用户剩余登录次数，该次数为 0 时表示此参数无效。

4.2 设置登录相关参数结构体 NET_PARAM

表4-2 NET_PARAM

选项	说明
结构体描述	设置登录时的相关参数结构体。

选项	说明
结构体	typedef struct { int nWaittime; int nConnectTime; int nConnectTryNum; int nSubConnectSpaceTime; int nGetDevInfoTime; int nConnectBufSize; int nGetConnInfoTime; int nSearchRecordTime; int nsubDisconnetTime; BYTE byNetType; BYTE byPlaybackBufSize; BYTE bDetectDisconnTime; BYTE bKeepLifeInterval; int nPicBufSize; BYTE bReserved[4]; } NET_PARAM;
成员	● nWaittime 等待超时时间（毫秒为单位），为 0 默认 5000ms。 ● nConnectTime 连接超时时间（毫秒为单位），为 0 默认 1500ms。 ● nConnectTryNum 连接尝试次数，为 0 默认 1 次。 ● nSubConnectSpaceTime 子连接之间的等待时间（毫秒为单位），为 0 默认 10ms。 ● nGetDevInfoTime 获取设备信息超时时间，为 0 默认 1000ms。 ● nConnectBufSize 每个连接接收数据缓冲大小（字节为单位），为 0 默认 250*1024。 ● nGetConnInfoTime 获取子连接信息超时时间（毫秒为单位），为 0 默认 1000ms。 ● nSearchRecordTime 按时间查询录像文件的超时时间(毫秒为单位)，为 0 默认为 3000ms。 ● nsubDisconnetTime 检测子链接断线等待时间（毫秒为单位），为 0 默认为 60000ms。 ● byNetType 网络类型，0-LAN，1-WAN。 ● byPlaybackBufSize 回放数据接收缓冲大小（M 为单位），为 0 默认为 4M。 ● bDetectDisconnTime 心跳检测断线时间（单位为秒），为 0 默认为 60s，最小时间为 2s。 ● bKeepLifeInterval 心跳包发送间隔（单位为秒），为 0 默认为 10s，最小间隔为 2s。 ● nPicBufSize 实时图片接收缓冲大小（字节为单位），为 0 默认为 2*1024*1024。 ● bReserved 保留字节。

4.3 设备信息扩展结构体 NET_DEVICEINFO_Ex

表4-3 NET_DEVICEINFO_Ex

选项	说明
结构体描述	设备信息扩展。
结构体	typedef struct { BYTE sSerialNumber[DH_SERIALNO_LEN]; int nAlarmInPortNum; int nAlarmOutPortNum; int nDiskNum; int nDVRType; int nChanNum; BYTE byLimitLoginTime; BYTE byLeftLogTimes; BYTE bReserved[2]; int nLockLeftTime; char Reserved[24]; } NET_DEVICEINFO_Ex, *LPNET_DEVICEINFO_Ex;
成员	● sSerialNumber 设备序列号。 ● nAlarmInPortNum DVR 报警输入个数。 ● nAlarmOutPortNum DVR 报警输出个数。 ● nDiskNum DVR 硬盘个数。 ● nDVRType DVR 类型，关于 DVR 类型的详细介绍，请参见“4.1 设备信息结构体 NET_DEVICEINFOI”。 ● nChanNum DVR 通道个数，登录成功时有效。 ● byLimitLoginTime 在线超时时间。为 0 表示不限制登录，非 0 表示限制的分钟数。 ● byLeftLogTimes 当登录失败原因为密码错误时，通过此参数通知用户，剩余登录次数，为 0 时表示此参数无效。 ● bReserved 保留字节，字节对齐。 ● nLockLeftTime 当登录失败，用户解锁剩余时间（秒数），-1 表示设备未设置该参数。 ● Reserved 保留字节。

4.4 登录接口入参结构体

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY

表4-4 NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY

选项	说明
结构体描述	CLIENT_LoginWithHighLevelSecurity 输入参数。
结构体	typedef struct tag NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY { DWORD dwSize; char szIP[64]; int nPort; char szUserName[64]; char szPassword[64]; EM_LOGIN_SPAC_CAP_TYPE emSpecCap; BYTE byReserved[4]; void* pCapParam; } NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY;
成员	● dwSize 结构体大小，使用时需赋值为 sizeof(NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY)。 ● szIp 设备 ip。 ● nPort 登录端口。 ● szUserName 用户名。 ● szPassword 密码。 ● emSpecCap 登录模式，设备支持的能力，具体模式请参见 EM_LOGIN_SPAC_CAP_TYPE 枚举说明。 ● byReserved 字节对齐。 ● pCapParam 对 emSpecCap 的补充参数，跟 emSpecCap 匹配使用，具体内容请参见 EM_LOGIN_SPAC_CAP_TYPE 枚举说明，若对应的 emSpecCap 值没有相应的 pCapParam 参数说明，则输入 NULL。

4.5 登录接口出参结构体

NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY

表4-5 NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY

选项	说明
结构体描述	CLIENT_LoginWithHighLevelSecurity 输出参数。

选项	说明
结构体	typedef struct tagNET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY { DWORD dwSize; NET_DEVICEINFO_Ex stuDeviceInfo; int nError; BYTE byReserved[132]; }NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY;
成员	● dwSize 结构体大小，使用时需赋值为 sizeof(NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY)。 ● stuDeviceInfo 设备登录成功时，保存登录设备的部分信息；登录失败时，保存登录有关的部分信息，如剩余登录次数等，具体请参见 NET_DEVICEINFO 结构体说明。 ● nError （当函数返回成功时，该参数的值无意义），返回登录错误码，如下： ◇ 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：账号被锁定 ◇ 6：账号被列入黑名单 ◇ 7：系统忙，资源不足 ◇ 8：子连接失败 ◇ 9：主连接失败 ◇ 10：超过最大连接数 ● byReserved 预留字段。

4.6 搜索接口入参结构体 NET_IN_STARTSERACH_DEVICE

表4-6 NET_IN_STARTSERACH_DEVICE

选项	说明
结构体描述	CLIENT_StartSearchDevicesEx 输入参数。
结构体	typedef struct tagNET_IN_STARTSERACH_DEVICE { DWORD dwSize; char szLocalIp[MAX_LOCAL_IP_LEN]; fSearchDevicesCBEx cbSearchDevices; void* pUserData; EM_SEND_SEARCH_TYPE emSendType; }NET_IN_STARTSERACH_DEVICE;

选项	说明
成员	• dwSize 结构体大小，使用时需赋值为 sizeof(NET_IN_STARTSERACH_DEVICE)。 • szLocalIp 发起搜索的本地 IP 可不输入，默认为 NULL。 • cbSearchDevices 设备信息回调函数当有设备响应包回复过来时，NetSDK 将响应包解析成有效的信息，通过回调函数通知用户，具体请参见 fSearchDevicesCBEx 回调函数说明。回调函数不可为空。 • pUserData 用户自定义数据，NetSDK 通过搜索设备回调函数 fSearchDevicesCB 将该数据返回给用户，以便用户后续操作。 • emSendType 搜索类型枚举，包含组播和广播，具体请参见 EM_SEND_SEARCH_TYPE 的枚举定义。

4.7 搜索接口出参结构体 NET_OUT_STARTSERACH_DEVICE

表4-7 NET_OUT_STARTSERACH_DEVICE

选项	说明
结构体描述	CLIENT_StartSearchDevicesEx 输出参数。
结构体	typedef struct tagNET_OUT_STARTSERACH_DEVICE { DWORD dwSize; }NET_OUT_STARTSERACH_DEVICE;
成员	dwSize：结构体大小，使用时需赋值为 sizeof(NET_OUT_STARTSERACH_DEVICE)。

4.8 视频数据帧信息结构体 tagVideoFrameParam

表4-8 tagVideoFrameParam

选项	说明
结构体描述	回调视频数据帧的帧参数结构体
结构体	typedef struct _tagVideoFrameParam { BYTE encode; BYTE frametype; BYTE format; BYTE size; DWORD fourcc; WORD width; WORD height; NET_TIME struTime; } tagVideoFrameParam;

选项	说明
成员	● encode：编码类型。 不同的值对应不同的编码类型，如下： ◇ 1：MPEG4 编码 ◇ 2：H264 编码 ◇ 3：ADI H264 编码 ◇ 4：标准 H264 编码 ● Frametype：帧类型。 不同的值对应不同的帧类型，如下： ◇ 0：I 帧 ◇ 1：P 帧 ◇ 2：B 帧 ● format 视频制式。 不同的值对应不同的视频制式，如下： ◇ 0：PAL ◇ 1：NTSC ● Size：分辨率。 不同的值对应不同的分辨率，如下： ◇ 0：CIF ◇ 1：HD1 ◇ 2：CIF ◇ 3：D1 ◇ 4：VGA ◇ 5：QCIF ◇ 6：QVGA ◇ 7：SVCD ◇ 8：QQVGA ◇ 9：SVGA ◇ 10：XVGA ◇ 11：WXGA ◇ 12：SXGA ◇ 13：WSXGA ◇ 14：UXGA ◇ 15：WUXGA ◇ 16：LFT ◇ 17：720 ◇ 18：1080 ● fourcc 如果是 H264 编码则总为 0，否则值为*(DWORD*)"DIVX"，即 0x58564944 ● width：宽，单位是像素，当 size=255 时有效 ● height：高，单位是像素，当 size=255 时有效 ● struTime：时间信息 具体请参见“4.10NET_TIME 结构体说明”

4.9 音频数据帧信息结构体 tagCBPCMDataParam

表4-9 tagCBPCMDataParam

选项	说明
结构体描述	回调音频数据帧的帧参数结构体

选项	说明
结构体	typedef struct _tagCBPCMDataParam { BYTE channels; BYTE samples; BYTE depth; BYTE param1; DWORD reserved; } tagCBPCMDataParam;
成员	● channels：声道数 ● samples：采样率 不同的值表示不同的采样率，如下： ◇ 0：8000 ◇ 1：11025 ◇ 2：16000 ◇ 3：22050 ◇ 4：32000 ◇ 5：44100 ◇ 6：48000 ● depth：采样深度取值 8 或者 16 等 ● param1：音频数据类型 ◇ 0：指示无符号 ◇ 1：指示有符号 ● reserved 保留

4.10 时间结构体 NET_TIME

表4-10 NET_TIME

选项	说明
结构体描述	时间结构体，精确到秒
结构体	typedef struct { DWORD dwYear; DWORD dwMonth; DWORD dwDay; DWORD dwHour; DWORD dwMinute; DWORD dwSecond; } NET_TIME,*LPNET_TIME;
成员	● dwYear：年 ● dwMonth：月 ● dwDay：日 ● dwHour：时 ● dwMinute：分 ● dwSecond：秒

4.11 录像文件信息结构体 NET_RECORDFILE_INFO

表4-11 NET_RECORDFILE_INFO

选项	说明
结构体描述	录像文件信息结构体
结构体	typedef struct { unsigned int ch; char filename[124]; unsigned int framenum; unsigned int size; NET_TIME starttime; NET_TIME endtime; unsigned int driveno; unsigned int startcluster; BYTE nRecordFileType; BYTE bImportantRecID; BYTE bHint; BYTE bRecType; } NET_RECORDFILE_INFO, *LPNET_RECORDFILE_INFO;

选项	说明
成员	● ch 通道号 ● filename 文件名 ● framenum 文件总帧数 ● size 文件长度 ● starttime 开始时间 ● endtime 结束时间 ● driveno 磁盘号（区分网络录像和本地录像的类型，0～127 表示本地录像，其中 64 表示“光盘 1”，128 表示“网络录像”） ● startcluster 起始簇号 ● nRecordFileType 录象文件类型 ◇ 0：普通录象 ◇ 1：报警录象 ◇ 2：移动检测 ◇ 3：卡号录象 ◇ 4：图片 ◇ 5：智能录像 ● bImportantRecID 是否为重要录像的标志 ◇ 0：普通录像 ◇ 1：重要录像 ● bHint 文件定位索引 （nRecordFileType==4<图片>时，bImportantRecID<<8 +bHint ，组成图片定位索引） ● bRecType 录像对应码流类型 ◇ 0：主码流录像 ◇ 1：辅码流 1 录像 ◇ 2：辅码流 2 录像 ◇ 3：辅码流 3 录像

4.12 云台能力集信息结构体

CFG_PTZ_PROTOCOL_CAPS_INFO

表4-12 CFG_PTZ_PROTOCOL_CAPS_INFO

选项	说明
结构体描述	云台能力集信息结构体

选项	说明
结构体	typedef struct tagCFG_PTZ_PROTOCOL_CAPS_INFO { int nStructSize; BOOL bPan; BOOL bTile; BOOL bZoom; BOOL bIris; BOOL bPreset; BOOL bRemovePreset; BOOL bTour; BOOL bRemoveTour; BOOL bPattern; BOOL bAutoPan; BOOL bAutoScan; BOOL bAux; BOOL bAlarm; BOOL bLight; BOOL bWiper; BOOL bFlip; BOOL bMenu; BOOL bMoveRelatively; BOOL bMoveAbsolutely; BOOL bReset; BOOL bGetStatus; BOOL bSupportLimit; BOOL bPtzDevice; BOOL bIsSupportViewRange; WORD wCamAddrMin; WORD wCamAddrMax; WORD wMonAddrMin; WORD wMonAddrMax; WORD wPresetMin; WORD wPresetMax; WORD wTourMin; WORD wTourMax; WORD wPatternMin; WORD wPatternMax; WORD wTileSpeedMin; WORD wTileSpeedMax; WORD wPanSpeedMin; WORD wPanSpeedMax; WORD wAutoScanMin; WORD wAutoScanMax; WORD wAuxMin; WORD wAuxMax; DWORD dwInterval; DWORD dwType; DWORD dwAlarmLen; DWORD dwNearLightNumber; DWORD dwFarLightNumber; DWORD dwSupportViewRangeType; DWORD dwSupportFocusMode; char szName[MAX_PROTOCOL_NAME_LEN]; char szAuxs[CFG_COMMON_STRING_32][CFG_COMMON_STRING_32];

选项	说明
成员	• nStructSize：赋值为 sizeof(CFG_PTZ_PROTOCOL_CAPS_INFO) • bPan：是否支持云台水平摆动 • bTile：是否支持云台垂直摆动 • bZoom：是否支持云台变倍 • bIris：是否支持云台光圈调节 • bPreset：是否支持预置点 • bRemovePreset：是否支持清除预置点 • bTour：是否支持自动巡航线路 • bRemoveTour：是否支持清除巡航 • bPattern：是否支持轨迹线路 • bAutoPan：是否支持自动水平摆动 • bAutoScan：是否支持自动扫描 • bAux：是否支持辅助功能 • bAlarm：是否支持报警功能 • bLight：是否支持灯光，内容请参见"stuPtzLightingControl"成员说明 • bWiper：是否支持雨刷 • bFlip：是否支持镜头翻转 • bMenu：是否支持云台内置菜单 • bMoveRelatively：是否支持云台按相对坐标定位 • bMoveAbsolutely：是否支持云台按绝对坐标定位 • bReset：是否支持云台复位 • bGetStatus：是否支持获取云台运动状态及方位坐标 • bSupportLimit：是否支持限位 • bPtzDevice：是否支持云台设备 • bIsSupportViewRange：是否支持云台可视域 • wCamAddrMin：通道地址的最小值 • wCamAddrMax：通道地址的最大值 • wMonAddrMin：预览地址的最小值 • wMonAddrMax：预览地址的最大值 • wPresetMin：预置点的最小值 • wPresetMax：预置点的最大值 • wTourMin：自动巡航线路的最小值 • wTourMax：自动巡航线路的最大值 • wPatternMin：轨迹线路的最小值 • wPatternMax：轨迹线路的最大值 • wTileSpeedMin：垂直速度的最小值 • wTileSpeedMax：垂直速度的最大值 • wPanSpeedMin：水平速度的最小值 • wPanSpeedMax：水平速度的最大值 • wAutoScanMin：自动扫描的最小值 • wAutoScanMax：自动扫描的最大值 • wAuxMin：辅助功能的最小值 • wAuxMax：辅助功能的最大值 • dwInterval：发送命令的时间间隔 • dwType：协议的类型，0-本地云台，1-远程云台 • dwAlarmLen：协议的报警长度 • dwNearLightNumber：近光灯组数量，0~4，为 0 时表示不支持 • dwFarLightNumber：远光灯组数量，0~4，为 0 时表示不支持 • dwSupportViewRangeType 支持的可视域数据获取方式掩码，从低位到高位依次数，目前支持 第 1 位：为 1 表示支持"ElectronicCompass" 电子罗盘方式。 • dwSupportFocusMode：支持的焦距模式掩码，从低位到高位依次数，具体请参见 EM_SUPPORT_FOCUS_MODE 枚举说明。 • szName 操作的协议名

选项	说明
	● szAuxs 云台辅助功能名称列表 ● stuPtzMotionRange 云台转动角度范围，单位：度，具体参见 CFG_PTZ_MOTION_RANGE 结构体说明 ● stuPtzLightingControl 灯光控制内容，具体参见 CFG_PTZ_LIGHTING_CONTROL 结构体说明 ● bSupportPresetTimeSection 是否支持预置点时间段配置的功能 ● bFocus 是否支持云台变焦

4.13 云台转动角度范围结构体 CFG_PTZ_MOTION_RANGE

表4-13 CFG_PTZ_MOTION_RANGE

选项	说明
结构体描述	云台转动角度范围结构体
结构体	typedef struct tagCFG_PTZ_MOTION_RANGE { int nHorizontalAngleMin; int nHorizontalAngleMax; int nVerticalAngleMin; int nVerticalAngleMax; }CFG_PTZ_MOTION_RANGE;
成员	● nHorizontalAngleMin：水平角度范围最小值，单位：度 ● nHorizontalAngleMax：水平角度范围最大值，单位：度 ● nVerticalAngleMin：垂直角度范围最小值，单位：度 ● nVerticalAngleMax：垂直角度范围最大值，单位：度

4.14 灯光控制内容结构体 CFG_PTZ_LIGHTING_CONTROL

表4-14 CFG_PTZ_LIGHTING_CONTROL

选项	说明
结构体描述	灯光控制内容结构体
结构体	typedef struct tagCFG_PTZ_LIGHTING_CONTROL { char szMode[CFG_COMMON_STRING_32]; DWORD dwNearLightNumber; DWORD dwFarLightNumber; }CFG_PTZ_LIGHTING_CONTROL;
成员	● szMode：手动灯光控制模式 ◇ "on-off"：直接开关模式, ◇ "adjustLight"：手动调节亮度模式 ● dwNearLightNumber：近光灯组数量 ● dwFarLightNumber：远光灯组数量

4.15 设备支持的语音对讲类型 DHDEV_TALKFORMAT_LIST

表4-15 DHDEV_TALKFORMAT_LIST

选项	说明
结构体描述	设备支持的语音对讲类型
结构体	typedef struct { int nSupportNum; DHDEV_TALKDECODE_INFO type[64]; char reserved[64]; } DHDEV_TALKFORMAT_LIST;
成员	● nSupportNum 支持个数 ● type 编码类型 具体参见 DHDEV_TALKDECODE_INFO 结构体说明 ● reserved 保留字节

4.16 语音编码信息结构体 DHDEV_TALKDECODE_INFO

表4-16 DHDEV_TALKDECODE_INFO

选项	说明
结构体描述	语音编码信息
结构体	typedef struct { DH_TALK_CODING_TYPE encodeType; int nAudioBit; DWORD dwSampleRate; int nPacketPeriod; char reserved[60]; } DHDEV_TALKDECODE_INFO;
成员	● encodeType：编码类型 具体请参见 DH_TALK_CODING_TYPE 枚举说明 ● nAudioBit：位数，如 8 或 16 ● dwSampleRate：采样率，如 8000 或 16000 ● nPacketPeriod：打包周期，单位 ms ● reserved：保留字节

4.17 系统信息结构体 DHDEV_SYSTEM_ATTR_CFG

表4-17 DHDEV_SYSTEM_ATTR_CFG

选项	说明
结构体描述	系统信息

选项	说明
结构体	typedef struct { DWORD dwSize; DH_VERSION_INFO stVersion; DH_DSP_ENCODECAP stDspEncodeCap; BYTE szDevSerialNo[DH_DEV_SERIALNO_LEN]; BYTE byDevType; BYTE szDevType[DH_DEV_TYPE_LEN]; BYTE byVideoCaptureNum; BYTE byAudioCaptureNum; BYTE byTalkInChanNum; BYTE byTalkOutChanNum; BYTE byDecodeChanNum; BYTE byAlarmInNum; BYTE byAlarmOutNum; BYTE byNetIONum; BYTE byUsbIONum; BYTE byIdeIONum; BYTE byComIONum; BYTE byLPTIONum; BYTE byVgaIONum; BYTE byIdeControlNum; BYTE byIdeControlType; BYTE byCapability; BYTE byMatrixOutNum; BYTE byOverWrite; BYTE byRecordLen; BYTE byDSTEnable; WORD wDevNo; BYTE byVideoStandard; BYTE byDateFormat; BYTE byDateSprtr; BYTE byTimeFmt; BYTE byLanguage; } DHDEV_SYSTEM_ATTR_CFG, *LPDHDEV_SYSTEM_ATTR_CFG;

选项	说明
成员	● dwSize：结构体大小 赋值为 sizeof(DHDEV_SYSTEM_ATTR_CFG) /* 下面是设备的只读部分 */ ● stVersion：设备版本信息 具体请参见 DH_VERSION_INFO 结构体说明 ● stDspEncodeCap：DSP 能力描述 具体请参见 DH_DSP_ENCODECAP 结构体说明 ● szDevSerialNo：设备序列号 ● byDevType：设备类型 具体请参见 NET_DEVICE_TYPE 枚举说明 ● szDevType：设备详细型号，字符串格式，可能为空 ● byVideoCaptureNum：视频口数量 ● byAudioCaptureNum：音频口数量 ● byTalkInChanNum：对讲输入接口数量 ● byTalkOutChanNum：对讲输出接口数量 ● byDecodeChanNum：解码接口数量 ● byAlarmInNum：报警输入口数 ● byAlarmOutNum：报警输出口数 ● byNetIONum：网络口数 ● byUsbIONum：USB 口数量 ● byIdeIONum：IDE 数量 ● byComIONum：串口数量 ● byLPTIONum：并口数量 ● byVgaIONum：VGA 接口数量 ● byIdeControlNum：IDE 控制数量 ● byIdeControlType：IDE 控制类型 ● byCapability：设备能力，扩展描述 ● byMatrixOutNum：视频矩阵输出口数 /* 下面是设备的可写部分 */ ● byOverWrite：硬盘满处理方式(1：覆盖:0：停止) ● byRecordLen：录像打包长度 ● byDSTEnable：是否实行夏令时 1-实行 0-不实行 ● wDevNo：设备编号，用于遥控 ● byVideoStandard：视频制式：0-PAL，1-NTSC ● byDateFormat：日期格式 ● byDateSprtr：日期分割符(0："."，1："-"，2："/") ● byTimeFmt：时间格式 （0～24 小时，1～12 小时） ● byLanguage:语言类型,具体请参见 DH_LANGUAGE_TYPE 枚举说明。

4.18 对讲参数结构体 NET_SPEAK_PARAM

表4-18 NET_SPEAK_PARAM

选项	说明
结构体描述	对讲参数结构体

选项	说明
结构体	typedef struct __NET_SPEAK_PARAM { DWORD dwSize; int nMode; int nSpeakerChannel; BOOL bEnableWait; } NET_SPEAK_PARAM;
成员	● dwSize 结构体大小，赋值为 sizeof(NET_SPEAK_PARAM) ● nMode 模式类型，0：对讲（默认模式），1：喊话；从喊话切换到对讲要重新设置。 ● nSpeakerChannel 扬声器通道号，喊话时有效。 ● bEnableWait 开启对讲时是否等待设备的响应，默认 FALSE。TRUE：等待；FALSE：不等待，超时时间由 CLIENT_SetNetworkParam 设置，对应 NET_PARAM 的 nWaittime 字段。

4.19 语音对讲的转发模式结构体 NET_TALK_TRANSFER_PARAM

表4-19 NET_TALK_TRANSFER_PARAM

选项	说明
结构体描述	是否开启语音对讲的转发模式
结构体	typedef struct tagNET_TALK_TRANSFER_PARAM { DWORD dwSize; BOOL bTransfer; }NET_TALK_TRANSFER_PARAM;
成员	● dwSize 结构体的大小，赋值为 sizeof(NET_TALK_TRANSFER_PARAM) ● bTransfer 是否开启语音对讲转发模式，TRUE：开启转发，FALSE：关闭转发

4.20 设备搜索回调信息结构体 DEVICE_NET_INFO_EX

表4-20 DEVICE_NET_INFO_EX

选项	说明
结构体描述	设备搜索回调信息结构体

选项	说明
结构体	typedef struct { int iIPVersion; char szIP[64]; int nPort; char szSubmask[64]; char szGateway[64]; char szMac[DH_MACADDR_LEN]; char szDeviceType[DH_DEV_TYPE_LEN]; BYTE byManuFactory; BYTE byDefinition; bool bDhcpEn; BYTE byReserved1; char verifyData[88]; char szSerialNo[DH_DEV_SERIALNO_LEN]; char szDevSoftVersion[DH_MAX_URL_LEN]; char szDetailType[DH_DEV_TYPE_LEN]; char szVendor[DH_MAX_STRING_LEN]; char szDevName[DH_MACHINE_NAME_NUM]; char szUserName[DH_USER_NAME_LENGTH_EX]; char szPassWord[DH_USER_NAME_LENGTH_EX]; unsigned short nHttpPort; WORD wVideoInputCh; WORD wRemoteVideoInputCh; WORD wVideoOutputCh; WORD wAlarmInputCh; WORD wAlarmOutputCh; char cReserved[244]; }DEVICE_NET_INFO_EX;

选项	说明
成员	● iIPVersion：ip 协议 ◇ 4：IPV4 ◇ 6：IPV6 ● szIP：ip 字符串形式 IPV4 形式如"192.168.0.1"，IPV6 形式如"2008::1/64" ● nPort：tcp 端口 ● szSubmask：子网掩码，IPV6 无子网掩码 ● szGateway：设备网关 ● szMac：设备 MAC 地址 ● szDeviceType：设备类型 ● byManuFactory：目标设备的生产厂商，具体请参见 EM_IPC_TYPE 枚举说明 ● byDefinition：1-标清 2-高清 ● bDhcpEn：DHCP 使能状态，true-开，false-关 ● byReserved1：字节对齐 ● verifyData：校验数据，通过异步搜索回调获取(在修改设备 IP 时会用此信息进行校验) ● szSerialNo：序列号 ● szDevSoftVersion：设备软件版本号 ● szDetailType：设备型号 ● szVendor：OEM 客户类型 ● szDevName：设备名称 ● szUserName：登录设备用户名（在修改设备 IP 时需要填写） ● szPassWord：登路设备密码（在修改设备 IP 时需要填写） ● nHttpPort：HTTP 服务端口号 ● wVideoInputCh：视频输入通道数 ● wRemoteVideoInputCh：远程视频输入通道数 ● wVideoOutputCh：视频输出通道数 ● wAlarmInputCh：报警输入通道数 ● wAlarmOutputCh：报警输出通道数 ● cReserved：保留字节

4.21 手动抓拍入参结构体 MANUAL_SNAP_PARAMETER

表4-21 MANUAL_SNAP_PARAMETER

选项	说明
结构体描述	手动抓拍入参结构体
结构体	typedef struct _MANUAL_SNAP_PARAMETER{ int nChannel; BYTE bySequence[64]; BYTE byReserved[60]; }MANUAL_SNAP_PARAMETER;
成员	● nChannel 抓图通道，从 0 开始 ● bySequence 抓图序列号字符串，当相应的智能图片报警上报时会返回该字段，同时触发多个交通手动抓拍事件时，用户可根据该字段做一一对应 ● byReserved 保留字段

4.22 权限信息扩展结构体 OPR_RIGHT_EX

表4-22 OPR_RIGHT_EX

选项	说明
结构体描述	权限信息结构体
结构体	typedef struct _OPR_RIGHT_EX { DWORD dwID; char name[DH_RIGHT_NAME_LENGTH]; char memo[DH_MEMO_LENGTH]; } OPR_RIGHT_EX;
成员	● dwID：权限 ID，每个权限都有各自的 ID ● name：权限名称 ● memo：权限备注说明

4.23 权限信息结构体 OPR_RIGHT_NEW

表4-23 OPR_RIGHT_NEW

选项	说明
结构体描述	权限信息结构体
结构体	typedef struct _OPR_RIGHT_NEW { DWORD dwSize; DWORD dwID; char name[DH_RIGHT_NAME_LENGTH]; char memo[DH_MEMO_LENGTH]; } OPR_RIGHT_NEW;
成员	● dwSize：结构体大小，赋值为 sizeof(OPR_RIGHT_NEW) ● dwID：权限 ID，每个权限都有各自的 ID ● name：权限名称 ● memo：权限备注说明

4.24 设备通道数量信息结构体 NET_DEV_CHN_COUNT_INFO

表4-24 NET_DEV_CHN_COUNT_INFO

选项	说明
结构体描述	设备通道数量信息结构体
结构体	typedef struct tagNET_DEV_CHN_COUNT_INFO { DWORD dwSize; NET_CHN_COUNT_INFO stuVideoIn; NET_CHN_COUNT_INFO stuVideoOut; } NET_DEV_CHN_COUNT_INFO;
成员	● dwSize 结构体大小，赋值为 sizeof(NET_DEV_CHN_COUNT_INFO) ● stuVideoIn 视频输入通道，具体请参见 NET_CHN_COUNT_INFO 结构体说明 ● stuVideoOut 视频输出通道，具体请参见 NET_CHN_COUNT_INFO 结构体说明

4.25 通道数量信息结构体 NET_CHN_COUNT_INFO

表4-25 NET_CHN_COUNT_INFO

选项	说明
结构体描述	通道数量信息结构体
结构体	typedef struct tagNET_CHN_COUNT_INFO { DWORD dwSize; int nMaxTotal; int nCurTotal; int nMaxLocal; int nCurLocal; int nMaxRemote; int nCurRemote; } NET_CHN_COUNT_INFO;
成员	● dwSize 结构体大小，赋值为 sizeof(NET_CHN_COUNT_INFO) ● nMaxTotal 设备总通道数(所有有效通道数之和) ● nCurTotal 已配置的通道数 ● nMaxLocal 最大本地通道数，含主板和可插拔子卡通道 ● nCurLocal 已配置本地通道数 ● nMaxRemote 最大远程通道数 ● nCurRemote 已配置远程通道数

4.26 获取抓图配置能力入参结构体 NET_IN_SNAP_CFG_CAPS

表4-26 NET_IN_SNAP_CFG_CAPS

选项	说明
结构体描述	获取抓图配置对应能力输入参数结构体
结构体	typedef struct tagNET_IN_SNAP_CFG_CAPS { int nChannelId; BYTE bReserved[1024]; }NET_IN_SNAP_CFG_CAPS;
成员	● nChannelId：通道号 ● bReserved：保留字节

4.27 获取抓图配置能力出参结构体 NET_OUT_SNAP_CFG_CAPS

表4-27 NET_OUT_SNAP_CFG_CAPS

选项	说明
结构体描述	获取抓图配置对应能力输出参数结构体
结构体	typedef struct tagNET_OUT_SNAP_CFG_CAPS { int nResolutionTypeNum; DH_RESOLUTION_INFO stuResolutionTypes[DH_MAX_CAPTURE_SIZE_NUM]; DWORD dwFramesPerSecNum; int nFramesPerSecList[DH_MAX_FPS_NUM]; DWORD dwQualityMun; DWORD nQualityList[DH_MAX_QUALITY_NUM]; DWORD dwMode; DWORD dwFormat; BYTE bReserved[2048]; } NET_OUT_SNAP_CFG_CAPS;
成员	• nResolutionTypeNum 支持的视频分辨率信息，与 stuResolutionTypes 匹配使用 • stuResolutionTypes 视频分辨率信息结构体，与 nResolutionTypeNum 匹配使用 • dwFramesPerSecNum 支持的帧率信息，与 nFramesPerSecList 匹配使用 • nFramesPerSecList 支持的帧率列表，与 dwFramesPerSecNum 匹配使用 • dwQualityMun 支持的画质信息，与 nQualityList 匹配使用 • nQualityList 支持的画质信息列表，与 dwQualityMun 匹配使用 • dwMode 模式,按位：第一位：定时；第二位：手动。 • dwFormat 图片格式模式,按位：第一位：bmp；第二位：jpg。 • bReserved 保留字节

4.28 图片分辨率结构体 DH_RESOLUTION_INFO

表4-28 DH_RESOLUTION_INFO

选项	说明
结构体描述	图片分辨率结构体
结构体	typedef struct { unsigned short snWidth; unsigned short snHight; }DH_RESOLUTION_INFO;

选项	说明
成员	● snWidth：宽度 ● snHight：高度

4.29 视频编码参数结构体 CFG_VIDEOENC_OPT

表4-29 CFG_VIDEOENC_OPT

选项	说明
结构体描述	视频编码参数结构体
结构体	typedef struct tagCFG_VIDEOENC_OPT { bool abVideoEnable; bool abAudioEnable; bool abSnapEnable; bool abAudioAdd; bool abAudioFormat; BOOL bVideoEnable; CFG_VIDEO_FORMAT stuVideoFormat; BOOL bAudioEnable; BOOL bSnapEnable; BOOL bAudioAddEnable; CFG_AUDIO_ENCODE_FORMAT stuAudioFormat; } CFG_VIDEOENC_OPT;

选项	说明
成员	● abVideoEnable 用于指示 bVideoEnable 字段是否有效 ◇ 获取时，用于标识是否支持视频使能 ◇ 设置时，用于指明是否修改视频使能 ● abAudioEnable 用于指示 bAudioEnable 字段是否有效 ◇ 获取时，用于标识是否支持音频使能 ◇ 设置时，用于指明是否修改音频使能 ● abSnapEnable 用于指示 bSnapEnable 字段是否有效 ◇ 获取时，用于标识是否支持定时抓图使能 ◇ 设置时，用于指明是否修改定时抓图使能 ● abAudioAdd 用于指示 bAudioAddEnable 字段是否有效 ◇ 获取时，用于标识是否支持音频叠加使能 ◇ 设置时，用于指明是否修改音频叠加使能 ● abAudioFormat 用于指示 stuAudioFormat 字段是否有效 ◇ 获取时，用于标识是否支持音频格式 ◇ 设置时，用于指明是否修改音频格式 ● bVideoEnable 视频使能，与 abVideoEnable 匹配使用 ● stuVideoFormat 视频格式，具体请参见 CFG_VIDEO_FORMAT 结构体说明 ● bAudioEnable 音频使能，与 abAudioEnable 匹配使用 ● bSnapEnable 定时抓图使能，与 abSnapEnable 匹配使用 ● bAudioAddEnable 音频叠加使能，与 abAudioAdd 匹配使用 ● stuAudioFormat 音频格式，与 abAudioFormat 匹配使用 具体请参见 CFG_AUDIO_ENCODE_FORMAT 结构体说明

4.30 视频格式结构体 CFG_VIDEO_FORMAT

表4-30 CFG_VIDEO_FORMAT

选项	说明
结构体描述	视频格式结构体

选项	说明
结构体	typedef struct tagCFG_VIDEO_FORMAT { bool abCompression; bool abWidth; bool abHeight; bool abBitRateControl; bool abBitRate; bool abFrameRate; bool abIFrameInterval; bool abImageQuality; bool abFrameType; bool abProfile; CFG_VIDEO_COMPRESSION emCompression; int nWidth; int nHeight; CFG_BITRATE_CONTROL emBitRateControl; int nBitRate; float nFrameRate; int nIFrameInterval; CFG_IMAGE_QUALITY emImageQuality; int nFrameType; CFG_H264_PROFILE_RANK emProfile; } CFG_VIDEO_FORMAT;

选项	说明
成员	• abCompression TRUE：emCompression 字段有效；FALSE：emCompression 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 • abWidth TRUE：nWidth 字段有效；FALSE：nWidth 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 • abHeight TRUE：nHeight 字段有效；FALSE：nHeight 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 • abBitRateControl TRUE：emBitRateControl 字段有效；FALSE：emBitRateControl 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 • abBitRate TRUE：nBitRate 字段有效；FALSE：nBitRate 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 • abFrameRate TRUE：nFrameRate 字段有效；FALSE：nFrameRate 字段无效。 此字段为只读字段，以获取时为准，不建议用户修改。 • abIFrameInterval TRUE：nIFrameInterval 字段有效；FALSE：nIFrameInterval 字段无效。 此字段为只读字段，以获取时为准，不建议用户修改。 • abImageQuality TRUE：emImageQuality 字段有效；FALSE：emImageQuality 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 • abFrameType TRUE：nFrameType 字段有效；FALSE：nFrameType 字段无效 此字段为只读字段，以获取时为准，不建议用户修改 • abProfile TRUE：emProfile 字段有效；FALSE：emProfile 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 • emCompression 视频压缩格式，该字段是否有效由 abCompression 决定。具体请参见 G_VIDEO_COMPRESSION 枚举说明。 • nWidth 视频宽度，该字段是否有效由 abWidth 决定。 • nHeight 视频高度，该字段是否有效由 abHeight 决定。 • emBitRateControl 码流控制模式，该字段是否有效由 abBitRateControl 决定。具体请参见 CFG_BITRATE_CONTROL 枚举说明。 • nBitRate 视频码流(kbps)，该字段是否有效由 abBitRate 决定。 • nFrameRate 视频帧率，该字段是否有效由 abFrameRate 决定。 • nIFrameInterval I 帧间隔(1-100)，比如 50 表示每 49 个 B 帧或 P 帧，设置一个 I 帧。该字段是否有效由 abIFrameInterval 决定。 • emImageQuality 图像质量，该字段是否有效由 abImageQuality 决定。具体请参见 CFG_IMAGE_QUALITY 枚举说明。 • nFrameType 打包模式，0－DHAV，1－"PS"。该字段是否有效由 abFrameType 决定。 • emProfile H.264 编码级别，该字段是否有效由 abProfile 决定。具体请参见 CFG_H264_PROFILE_RANK 枚举说明。

4.31 音频格式结构体 CFG_AUDIO_ENCODE_FORMAT

表4-31 CFG_AUDIO_ENCODE_FORMAT

选项	说明
结构体描述	音频格式结构体
结构体	typedef struct tagCFG_AUDIO_FORMAT { bool abCompression; bool abDepth; bool abFrequency; bool abMode; bool abFrameType; bool abPacketPeriod; CFG_AUDIO_FORMAT emCompression; AV_int32 nDepth; AV_int32 nFrequency; AV_int32 nMode; AV_int32 nFrameType; AV_int32 nPacketPeriod; } CFG_AUDIO_ENCODE_FORMAT;

选项	说明
成员	● abCompression TRUE：emCompression 字段有效；FALSE：emCompression 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 ● abDepth TRUE：nDepth 字段有效；FALSE：nDepth 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 ● abFrequency TRUE：nFrequency 字段有效；FALSE：nFrequency 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 ● abMode TRUE：nMode 字段有效；FALSE：nMode 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 ● abFrameType TRUE：nFrameType 字段有效；FALSE：nFrameType 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 ● abPacketPeriod TRUE：nPacketPeriod 字段有效；FALSE：nPacketPeriod 字段无效。此字段为只读字段，以获取时为准，不建议用户修改。 ● emCompression 音频压缩模式。该字段是否有效由 abCompression 决定，具体请参见 CFG_AUDIO_FORMAT 枚举说明。 ● nDepth 音频采样深度。该字段是否有效由 abDepth 决定。 ● nFrequency 音频采样频率。该字段是否有效由 abFrequency 决定。 ● nMode 音频编码模式。该字段是否有效由 abMode 决定。 ● nFrameType 音频打包模式， 0-DHAV, 1-PS。该字段是否有效由 abFrameType 决定。 ● nPacketPeriod 音频打包周期，单位毫秒。该字段是否有效由 abPacketPeriod 决定。

4.32 多区域遮挡配置结构体 CFG_VIDEO_COVER

表4-32 CFG_VIDEO_COVER

选项	说明
结构体描述	多区域遮挡配置结构体
结构体	typedef struct tagCFG_VIDEO_COVER { int nTotalBlocks; int nCurBlocks; CFG_COVER_INFO stuCoverBlock[MAX_VIDEO_COVER_NUM]; } CFG_VIDEO_COVER;
成员	● nTotalBlocks：支持的遮挡块数 ● nCurBlocks：已设置的块数 ● stuCoverBlock：覆盖的区域。 具体请参见 CFG_COVER_INFO 结构体说明。

4.33 遮挡信息结构体 CFG_COVER_INFO

表4-33 CFG_COVER_INFO

选项	说明
结构体描述	遮挡信息结构体
结构体	typedef struct tagCFG_COVER_INFO { bool abBlockType; bool abEncodeBlend; bool abPreviewBlend; CFG_RECT stuRect; CFG_RGBA stuColor; int nBlockType; int nEncodeBlend; int nPreviewBlend; } CFG_COVER_INFO;
成员	● abBlockType TRUE：nBlockType 字段有效；FALSE：nBlockType 字段无效 此字段为只读字段，以获取时为准，不建议用户修改 ● abEncodeBlend TRUE：nEncodeBlend 字段有效；FALSE：nEncodeBlend 字段无效。 此字段为只读字段，以获取时为准，不建议用户修改 ● abPreviewBlend TRUE：nPreviewBlend 字段有效；FALSE：nPreviewBlend 字段无效。 此字段为只读字段，以获取时为准，不建议用户修改 ● stuRect 覆盖的区域坐标。具体参见 CFG_RECT 结构体说明 ● stuColor 覆盖的颜色。具体参见 CFG_RGBA 结构体说明 ● nBlockType 覆盖方式；0－黑块，1－马赛克。该字段是否有效由 abBlockType 决定 ● nEncodeBlend 编码级遮挡；1－生效，0－不生效。该字段是否有效由 abEncodeBlend 决定 ● nPreviewBlend 预览遮挡；1－生效，0－不生效。该字段是否有效由 abPreviewBlend 决定

4.34 区域信息结构体 CFG_RECT

表4-34 CFG_RECT

选项	说明
结构体描述	区域信息结构体

选项	说明
结构体	typedef struct tagCFG_RECT { int nLeft; int nTop; int nRight; int nBottom; } CFG_RECT;
成员	● nLeft：区域左侧 ● nTop：区域顶部 ● nRight：区域右侧 ● nBottom：区域底部
结构体描述	RGBA 信息结构体
结构体	typedef struct tagCFG_RGBA { int nRed; int nGreen; int nBlue; int nAlpha; } CFG_RGBA;
成员	● nRed：红 ● nGreen：绿 ● nBlue：蓝 ● nAlpha：透明度

4.35 图像通道属性结构体 CFG_ENCODE_INFO

表4-35 CFG_ENCODE_INFO

选项	说明
结构体描述	图像通道属性信息结构体
结构体	typedef struct tagCFG_ENCODE_INFO { int nChannelID; char szChnName[MAX_CHANNELNAME_LEN]; CFG_VIDEOENC_OPT stuMainStream[MAX_VIDEOSTREAM_NUM]; CFG_VIDEOENC_OPT stuExtraStream[MAX_VIDEOSTREAM_NUM]; CFG_VIDEOENC_OPT stuSnapFormat[MAX_VIDEOSTREAM_NUM]; DWORD dwCoverAbilityMask; DWORD dwCoverEnableMask; CFG_VIDEO_COVER stuVideoCover; CFG_OSD_INFO stuChnTitle; CFG_OSD_INFO stuTimeTitle; CFG_COLOR_INFO stuVideoColor; CFG_AUDIO_FORMAT emAudioFormat; int nProtocolVer; } CFG_ENCODE_INFO;

选项	说明
成员	● nChannelID：通道号（从 0 开始）。 获取时，该字段有效；设置时，该字段无效 ● szChnName：无效字段 ● stuMainStream：主码流属性信息 ◇ stuMainStream[0]－主码流普通录像属性信息 ◇ stuMainStream[1]－主码流动检录像属性信息 ◇ stuMainStream[2]－主码流报警录像属性信息 具体请参见 CFG_VIDEOENC_OPT 结构体说明 ● stuExtraStream：辅码流属性信息 ◇ stuExtraStream[0]－辅码流普通录像属性信息 ◇ stuExtraStream[1]－辅码流动检录像属性信息 ◇ stuExtraStream[2]－辅码流报警录像属性信息 具体请参见 CFG_VIDEOENC_OPT 结构体说明 ● stuSnapFormat：抓图属性信息 ◇ stuSnapFormat[0]－普通抓图属性信息 ◇ stuSnapFormat[1]－动检抓图属性信息 ◇ stuSnapFormat[2]－报警抓图属性信息 具体请参见 CFG_VIDEOENC_OPT 结构体说明 ● dwCoverAbilityMask：无效字段 ● dwCoverEnableMask：无效字段 ● stuVideoCover：无效字段 ● stuChnTitle：无效字段 ● stuTimeTitle：无效字段 ● stuVideoColor：无效字段 ● emAudioFormat：无效字段 ● nProtocolVer：协议版本号 只读。获取时，该字段有效；设置时，该字段无效。

4.36 抓图参数结构体 SNAP_PARAMS

表4-36 SNAP_PARAMS

选项	说明
结构体描述	抓图参数结构体
结构体	typedef struct _snap_param { unsigned int Channel; unsigned int Quality; unsigned int ImageSize; unsigned int mode; unsigned int InterSnap; unsigned int CmdSerial; unsigned int Reserved[4]; } SNAP_PARAMS, *LPSNAP_PARAMS;

选项	说明
成员	● Channel 抓图的通道 ● Quality 画质，数值范围 1～6，数值越高，画质越好 ● ImageSize 画面大小；0：QCIF,1：CIF,2：D1 ● mode 抓图模式 ◇ -1：表示停止抓图 ◇ 0：表示请求一帧 ◇ 1：表示定时发送请求 ◇ 2：表示连续请求 ● InterSnap 时间间隔，单位秒；若 mode=1 表示定时发送请求时，只有部分特殊设备(如：车载设备)支持通过该字段实现定时抓图时间间隔的配置，建议通过 CFG_CMD_ENCODE 配置的 stuSnapFormat[nSnapMode].stuVideoFormat.nFrameRate 字段实现相关功能 ● CmdSerial 抓图请求序列号，有效值范围 0~65535，超过范围会被截断为 unsigned short ● Reserved 保留字节

4.37 设备软件版本结构体 DH_VERSION_INFO

表4-37 DH_VERSION_INFO

选项	说明
结构体描述	设备软件版本信息,高 16 位表示主版本号,低 16 位表示次版本号
结构体	typedef struct { DWORD dwSoftwareVersion; DWORD dwSoftwareBuildDate; DWORD dwDspSoftwareVersion; DWORD dwDspSoftwareBuildDate; DWORD dwPanelVersion; DWORD dwPanelSoftwareBuildDate; DWORD dwHardwareVersion; DWORD dwHardwareDate; DWORD dwWebVersion; DWORD dwWebBuildDate; } DH_VERSION_INFO, *LPDH_VERSION_INFO;

选项	说明
成员	• dwSoftwareVersion：软件版本 • dwSoftwareBuildDate：软件版本编译日期 • dwDspSoftwareVersion：dsp 软件版本 • dwDspSoftwareBuildDate：dsp 软件版本编译日期 • dwPanelVersion：现在没有使用 • dwPanelSoftwareBuildDate：现在没有使用 • dwHardwareVersion：硬件版本 • dwHardwareDate：现在没有使用 • dwWebVersion：web 版本 • dwWebBuildDate：web 版本编译日期

4.38 开始实时预览转码入参 NET_IN_REALPLAY_BY_DATA_TYPE

表4-38 NET_IN_REALPLAY_BY_DATA_TYPE

选项	说明
结构体描述	CLIENT_RealPlayByDataType 入参
结构体	typedef struct tagNET_IN_REALPLAY_BY_DATA_TYPE { DWORD dwSize; int nChannelID; HWND hWnd; DH_RealPlayType rType; fRealDataCallBackEx cbRealData; EM_REAL_DATA_TYPE emDataType; LDWORD dwUser; const char* szSaveFileName; fRealDataCallBackEx2 cbRealDataEx; EM_AUDIO_DATA_TYPE emAudioType; fDataCallBackEx cbRealDataEx2; }NET_IN_REALPLAY_BY_DATA_TYPE;

选项	说明
成员	● dwSize 结构体大小 ● nChannelID 通道编号 ● hWnd 窗口句柄 ● rType 码流类型 ● cbRealData 数据回调函数 ● emDataType 回调的数据类型 ● szSaveFileName 转换后的文件名 ● cbRealDataEx 数据回调函数-扩展 ● emAudioType 音频格式 ● cbRealDataEx2 数据回调（扩展带时间戳，帧类型）

4.39 开始实时预览转码出参 NET_OUT_REALPLAY_BY_DATA_TYPE

表4-39 NET_OUT_REALPLAY_BY_DATA_TYPE

选项	说明
结构体描述	CLIENT_RealPlayByDataType 出参。
结构体	typedef struct tagNET_OUT_REALPLAY_BY_DATA_TYPE { DWORD dwSize; }NET_OUT_REALPLAY_BY_DATA_TYPE;
成员	dwSize：结构体大小，使用时需赋值为 sizeof(NET_OUT_REALPLAY_BY_DATA_TYPE)。

4.40 开始录像回放转码入参 NET_IN_PLAYBACK_BY_DATA_TYPE

表4-40 NET_IN_PLAYBACK_BY_DATA_TYPE

选项	说明
结构体描述	CLIENT_PlayBackByDataType 入参

选项	说明
结构体	typedef struct tagNET_IN_PLAYBACK_BY_DATA_TYPE { DWORD dwSize; int nChannelID; NET_TIME stStartTime; NET_TIME stStopTime; HWND hWnd; fDownLoadPosCallBack cbDownLoadPos; LDWORD dwPosUser; fDataCallBack fDownLoadDataCallBack; EM_REAL_DATA_TYPE emDataType; LDWORD dwDataUser; int nPlayDirection; EM_AUDIO_DATA_TYPE emAudioType; fDataCallBackEx fDownLoadDataCallBackEx; }NET_IN_PLAYBACK_BY_DATA_TYPE;
成员	● dwSize 结构体大小 ● nChannelID 通道编号 ● stStartTime 开始时间 ● stStopTime 结束时间 ● hWnd 窗口句柄 ● cbDownLoadPos 进度回调 ● dwPosUser 进度回调用户信息 ● fDownLoadDataCallBack 数据回调 ● emDataType 回调数据类型 ● dwDataUser 数据回调用户信息 ● nPlayDirection 播放方向,0:正放;1:倒放 ● emAudioType 音频类型 ● fDownLoadDataCallBackEx 数据回调（扩展带时间戳，帧类型）

4.41 开始录像回放转码出参 NET_OUT_PLAYBACK_BY_DATA_TYPE

表4-41 NET_OUT_PLAYBACK_BY_DATA_TYPE

选项	说明
结构体描述	CLIENT_PlayBackByDataType 出参。

选项	说明
结构体	typedef struct tagNET_OUT_PLAYBACK_BY_DATA_TYPE { DWORD dwSize; }NET_OUT_PLAYBACK_BY_DATA_TYPE;
成员	dwSize：结构体大小，使用时需赋值为 sizeof(NET_OUT_PLAYBACK_BY_DATA_TYPE)。

4.42 开始录像下载转码入参

NET_IN_DOWNLOAD_BY_DATA_TYPE

表4-42 NET_IN_DOWNLOAD_BY_DATA_TYPE

选项	说明
结构体描述	CLIENT_DownloadByDataType 入参
结构体	typedef struct tagNET_IN_DOWNLOAD_BY_DATA_TYPE { DWORD dwSize; int nChannelID; EM_QUERY_RECORD_TYPE emRecordType; char* szSavedFileName; NET_TIME stStartTime; NET_TIME stStopTime; fTimeDownLoadPosCallBack cbDownLoadPos; LDWORD dwPosUser; fDataCallBack fDownLoadDataCallBack; EM_REAL_DATA_TYPE emDataType; LDWORD dwDataUser; EM_AUDIO_DATA_TYPE emAudioType; }NET_IN_DOWNLOAD_BY_DATA_TYPE;

选项	说明
成员	● dwSize 结构体大小 ● nChannelID 通道编号 ● emRecordType 录像类型 ● szSavedFileName 下载的文件路径 ● stStartTime 开始时间 ● stStopTime 结束时间 ● cbDownLoadPos 进度回调 ● dwPosUser 进度回调用户信息 ● fDownLoadDataCallBack 数据回调 ● emDataType 回调的数据类型 ● dwDataUser 数据回调用户信息 ● emAudioType 音频类型

4.43 开始录像下载转码出参 NET_OUT_DOWNLOAD_BY_DATA_TYPE

表4-43 NET_OUT_DOWNLOAD_BY_DATA_TYPE

选项	说明
结构体描述	CLIENT_DownloadByDataType 出参。
结构体	typedef struct tagNET_OUT_DOWNLOAD_BY_DATA_TYPE { DWORD dwSize; }NET_OUT_DOWNLOAD_BY_DATA_TYPE;
成员	dwSize：结构体大小，使用时需赋值为 sizeof(NET_OUT_DOWNLOAD_BY_DATA_TYPE)。

4.44 DSP 能力描述结构体 DH_DSP_ENCODECAP

表4-44 DH_DSP_ENCODECAP

选项	说明
结构体描述	DSP 能力描述结构体。

选项	说明
结构体	typedef struct { DWORD dwVideoStandardMask; DWORD dwImageSizeMask; DWORD dwEncodeModeMask; DWORD dwStreamCap; DWORD dwImageSizeMask_Assi[8]; DWORD dwMaxEncodePower; WORD wMaxSupportChannel; WORD wChannelMaxSetSync; } DH_DSP_ENCODECAP, *LPDH_DSP_ENCODECAP;

选项	说明
成员	● dwVideoStandardMask 视频制式掩码，按位表示设备能够支持的视频制式. ● dwImageSizeMask 分辨率掩码，按位表示设备能够支持的分辨率设置，分辨率具体说明如下： ◇ 0：704*576(PAL) 704*480(NTSC) ◇ 1：352*576(PAL) 352*480(NTSC) ◇ 2：704*288(PAL) 704*240(NTSC) ◇ 3：352*288(PAL) 352*240(NTSC) ◇ 4：176*144(PAL) 176*120(NTSC) ◇ 5：640*480 ◇ 6：320*240 ◇ 7：480*480 ◇ 8：160*128 ◇ 9：800*592 ◇ 10：1024*768 ◇ 11：1280*800 ◇ 12：1600*1024 ◇ 13：1600*1200 ◇ 14：1920*1200 ◇ 15：240*192 ◇ 16：1280*720 ◇ 17：1920*1080 ◇ 18：1280*960 ◇ 19：1872*1408 ◇ 20：3744*1408 ◇ 21：2048*1536 ◇ 22：2432*2050 ◇ 23：1216*1024 ◇ 24：1408*1024 ◇ 25：3296*2472 ◇ 26：2560*1920(5M) ◇ 27：960*576(PAL) 960*480(NTSC) ◇ 28：960*720 ● dwEncodeModeMask 编码模式掩码，按位表示设备能够支持的编码模式设置. ● dwStreamCap 按位表示设备支持的多媒体功能： ◇ 第一位表示支持主码流 ◇ 第二位表示支持辅码流 1 ◇ 第三位表示支持辅码流 2 ◇ 第五位表示支持 jpg 抓图 ● dwImageSizeMask_Assi 表示主码流为各分辨率时，支持的辅码流分辨率掩码. ● dwMaxEncodePower DSP 支持的最高编码能力. ● wMaxSupportChannel 每块 DSP 支持最多输入视频通道数. ● wChannelMaxSetSync DSP 每通道的最大编码设置是否同步；0：不同步，1：同步

4.45 获取设备升级状态 DHDEV_UPGRADE_STATE_INFO

表4-45 DHDEV_UPGRADE_STATE_INFO

选项	说明
结构体描述	获取设备升级状态。

结构体	typedef struct tagDHDEV_UPGRADE_STATE_INFO
	{
	int nState; /// 0:None-没有检测到更新，默认状态;
	/// 1:Regular-一般升级;
	/// 2:Emergency-强制升级;
	/// 3:Upgrading-升级中
	char szOldVersion[UPDATE_VERSION_LEN]; /// 旧版本
	char szNewVersion[UPDATE_VERSION_LEN]; /// 新版本
	DWORD dwProgress; /// 升级进度
	int reserved[256]; /// 保留字段
	}DHDEV_UPGRADE_STATE_INFO;

成员	● nState 升级状态：0:None-没有检测到更新，默认状态; 1:Regular-一般升级; 2:Emergency-强制升级; 3:Upgrading-升级中. ● szOldVersion 旧版本号 ● szNewVersion 新版本号. ● dwProgress 升级进度

4.46 启动程序入参结构体 NET_IN_START_APP

表4-46 NET_IN_START_APP

选项	说明
结构体描述	CLIENT_StartApp 入参

结构体	typedef struct tagNET_IN_START_APP
	{
	DWORD dwSize; /// 结构体大小
	UINT nAppID; /// APP 的 ID
	char szAppName[128]; /// APP 的名称
	} NET_IN_START_APP;

成员	● dwSize 结构体大小 ● nAppID APP 的 ID ● szAppName APP 的名称

4.47 启动程序出参结构体 NET_OUT_START_APP

表4-47 NET_OUT_START_APP

选项	说明
结构体描述	CLIENT_StartApp 出参

选项	说明

结构体	typedef struct tagNET_OUT_START_APP
	{
	DWORD dwSize; /// 结构体大小
	} NET_OUT_START_APP;

成员	● dwSize 结构体大小

4.48 APP 查询配置信息结构体 CFG_DEV_DISPOSITION_INFO

表4-48 CFG_DEV_DISPOSITION_INFO

选项	说明
结构体描述	CLIENT_GetNewDevConfig 出参

结构体	typedef struct tagCFG_DEV_DISPOSITION_INFO
	{
	int nLocalNo; // 本机编号，主要用于遥控器区分不同设备 0~998
	char szMachineName[256]; // 机器名称或编号
	char szMachineAddress[256]; // 机器部署地点即道路编码
	char szMachineGroup[256]; // 机器分组或叫设备所属单位 默认为空，用户可以将不同的设备编为一组，便于管理，可重复。
	char szMachineID[64]; // 机器编号, 联网平台内唯一
	int nLockLoginTimes; // 登陆失败可尝试次数
	int nLoginFailLockTime; // 登陆失败锁定时间
	BOOL bLockLoginEnable; // 登陆失败可尝试次数使能
	CFG_DATA_TIME stuActivationTime; // 启动时间
	UINT nCheckDuration; // 清零周期，如果规定时间内登陆次数未超过可尝试次数，尝试次数清零
	BOOL bUseLocalPolicy; // 是否使用本地 GUI 锁定策略
	CFG_LOCAL_POLICY_INFO stuLocalPolicy; // 本地 GUI 锁定策略
	char szMachineName1[256]; // 机器名称或编号
	BYTE bReserved[508]; // 保留字节
	}CFG_DEV_DISPOSITION_INFO;

选项	说明
成员	● nLocalNo 本机编号 ● szMachineName 机器名称或编号 ● szMachineAddress 机器部署地点即道路编码 ● szMachineGroup 机器分组 ● szMachineID 机器编号 ● nLockLoginTimes 登陆失败可尝试次数 ● nLoginFailLockTime 登陆失败锁定时间 ● bLockLoginEnable 登陆失败可尝试次数使能 ● stuActivationTime 启动时间 ● nCheckDuration 清零周期 ● bUseLocalPolicy 是否使用本地 GUI 锁定策略 ● stuLocalPolicy 本地 GUI 锁定策略 ● szMachineName1 机器名称或编号

4.49 获取安装的应用列表入参结构体 NET_IN_GET_INSTALLED_APP_INFO

表4-49 NET_IN_GET_INSTALLED_APP_INFO

选项	说明
结构体描述	CLIENT_GetInstalledAppInfo 入参

结构体	typedef struct tagNET_IN_GET_INSTALLED_APP_INFO
	{
	DWORD dwSize; /// 结构体大小
	} NET_IN_GET_INSTALLED_APP_INFO;

成员	● dwSize 结构体大小

4.50 获取安装的应用列表出参结构体 NET_OUT_GET_INSTALLED_APP_INFO

表4-50 NET_OUT_GET_INSTALLED_APP_INFO

选项	说明
结构体描述	CLIENT_GetInstalledAppInfo 出参

选项	说明

结构体	typedef struct tagNET_OUT_GET_INSTALLED_APP_INFO
	{
	DWORD dwSize; /// 结构体大小
	UINT nListCount; /// 返回以安装的应用列表信息的数量
	NET_INSTALLED_APP_INFO stuAppInfoList[16]; /// 安装的应用列表信息
	} NET_OUT_GET_INSTALLED_APP_INFO;

成员	● dwSize
	结构体大小
	● nListCount
	安装的应用列表信息的数量
	● stuAppInfoList
	安装的应用列表信息
	///@brief 安装的应用信息
	typedef struct tagNET_INSTALLED_APP_INFO
	{
	char szAppName[128]; /// APP 的名称
	char szVersion[64]; /// APP 的版本
	char szExtend[64]; /// 扩展信息
	UINT nAppID; /// APP 的 ID
	EM_APP_DEBUG_STATE emAppDebugState; /// APP 的调试状态
	EM_APP_RUNNING_STATE emAppRunningState; ///APP 的运行状态
	EM_APP_LICENSE_STATE emAppLicenseState;/// APP 的 license 状态
	NET_APP_CAPS stuAppCaps; /// APP 的能力信息
	BYTE byReserved[256]; /// 保留字节
	} NET_INSTALLED_APP_INFO;

4.51 升级准备入参结构体 NET_IN_INSTALL_PREPAREEX

表4-51 NET_IN_INSTALL_PREPAREEX

选项	说明
结构体描述	CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_PREPAREEX 入参

结构体	typedef struct tagNET_IN_INSTALL_PREPAREEX
	{
	DWORD dwSize; /// 结构体大小
	char szAppName[32]; /// 升级的 app 名称
	EM_NET_NEXT_OPERATE emNextOperate; /// 接下来的操作
	UINT nTotalLength; /// 升级包大小
	BOOL bReliable; /// 升级可靠性
	}NET_IN_INSTALL_PREPAREEX;

选项	说明
成员	● dwSize 结构体大小 ● szAppName 升级的 app 名称 ● emNextOperate 接下来的操作 ● nTotalLength 升级包大小 ● bReliable 升级可靠性

4.52 升级准备出参结构体 NET_OUT_INSTALL_PREPAREEX

表4-52 NET_OUT_INSTALL_PREPAREEX

选项	说明
结构体描述	CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_PREPAREEX 出参

结构体	typedef struct tagNET_OUT_INSTALL_PREPAREEX
	{
	DWORD dwSize; /// 结构体大小
	}NET_OUT_INSTALL_PREPAREEX;

成员	● dwSize 结构体大小

4.53 推送升级数据入参结构体 NET_IN_INSTALL_APPEND_DATA

表4-53 NET_IN_INSTALL_APPEND_DATA

选项	说明
结构体描述	CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_APPEND_DATA 入参

结构体	typedef struct tagNET_IN_INSTALL_APPEND_DATA
	{
	DWORD dwSize; /// 结构体大小
	UINT nTotalLen; /// 升级文件总大小
	UINT nAppendLen; /// 本次发送的二进制长度
	BYTE* pAppendData; /// 本次发送的二进制数据 由用户申请释放
	}NET_IN_INSTALL_APPEND_DATA;

成员	● dwSize 结构体大小 ● nTotalLen 升级文件总大小 ● nAppendLen 本次发送的二进制长度 ● pAppendData 本次发送的二进制数据

4.54 推送升级数据出参结构体 NET_OUT_INSTALL_APPEND_DATA

表4-54 NET_OUT_INSTALL_APPEND_DATA

选项	说明
结构体描述	CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_APPEND_DATA 出参

结构体	typedef struct tagNET_OUT_INSTALL_APPEND_DATA
	{
	DWORD dwSize; /// 结构体大小
	}NET_OUT_INSTALL_APPEND_DATA;

成员	● dwSize 结构体大小

4.55 执行升级入参结构体 NET_IN_INSTALL_EXECUTE

表4-55 NET_IN_INSTALL_EXECUTE

选项	说明
结构体描述	CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_EXECUTE 入参

结构体	typedef struct tagNET_IN_INSTALL_EXECUTE
	{
	DWORD dwSize; /// 结构体大小
	BOOL bAutoReboot; /// 升级后是否自动重启
	}NET_IN_INSTALL_EXECUTE;

成员	● dwSize 结构体大小 ● bAutoReboot 升级后是否自动重启

4.56 执行升级出参结构体 NET_OUT_INSTALL_EXECUTE

表4-56 NET_OUT_INSTALL_EXECUTE

选项	说明
结构体描述	CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_EXECUTE 出参

结构体	typedef struct tagNET_OUT_INSTALL_EXECUTE
	{
	DWORD dwSize; /// 结构体大小
	}NET_OUT_INSTALL_EXECUTE;

成员	● dwSize 结构体大小

4.57 停止程序运行入参结构体 NET_IN_STOP_APP

表4-57 NET_IN_STOP_APP

选项	说明
结构体描述	CLIENT_StopApp 入参

选项	说明

结构体	typedef struct tagNET_IN_STOP_APP
	{
	DWORD dwSize; /// 结构体大小
	UINT nAppID; /// APP 的 ID
	char szAppName[128]; /// APP 的名称
	} NET_IN_STOP_APP;

成员	● dwSize 结构体大小 ● nAppID APP 的 ID ● szAppName APP 的名称

4.58 停止程序运行出参结构体 NET_OUT_STOP_APP

表4-58 NET_OUT_STOP_APP

选项	说明
结构体描述	CLIENT_StopApp 出参

结构体	typedef struct tagNET_OUT_STOP_APP
	{
	DWORD dwSize; /// 结构体大小
	} NET_OUT_STOP_APP;

成员	● dwSize 结构体大小

4.59 卸载程序入参结构体 NET_IN_INSTALL_EXECUTE

表4-59 NET_IN_INSTALL_EXECUTE

选项	说明
结构体描述	CLIENT_RemoveApp 入参

结构体	typedef struct tagNET_IN_REMOVE_APP
	{
	DWORD dwSize; /// 结构体大小
	UINT nAppID; /// APP 的 ID
	char szAppName[128]; /// APP 的名称
	} NET_IN_REMOVE_APP;

成员	● dwSize 结构体大小 ● nAppID APP 的 ID ● szAppName APP 的名称

4.60 卸载程序出参结构体 NET_OUT_INSTALL_EXECUTE

表4-60 NET_OUT_INSTALL_EXECUTE

选项	说明
结构体描述	CLIENT_RemoveApp 出参

选项	说明

结构体	typedef struct tagNET_OUT_REMOVE_APP
	{
	DWORD dwSize; /// 结构体大小
	} NET_OUT_REMOVE_APP;

成员	● dwSize 结构体大小

第 5 章 枚举定义

5.1 设备类型枚举 NET_DEVICE_TYPE

表5-1 NET_DEVICE_TYPE

选项	说明
枚举描述	设备类型枚举，用于对应不同的设备类型

选项	说明

枚举定义	typedef enum tagNET_DEVICE_TYPE
	{
	NET_PRODUCT_NONE = 0,
	NET_DVR_NONREALTIME_MACE, // 非实时 MACE
	NET_DVR_NONREALTIME, // 非实时
	NET_NVS_MPEG1, // 网络视频服务器
	NET_DVR_MPEG1_2, // MPEG1 二路录像机
	NET_DVR_MPEG1_8, // MPEG1 八路录像机
	NET_DVR_MPEG4_8, // MPEG4 八路录像机
	NET_DVR_MPEG4_16, // MPEG4 十六路录像机
	NET_DVR_MPEG4_SX2, // LB 系列录像机
	NET_DVR_MEPG4_ST2, // GB 系列录像机
	NET_DVR_MEPG4_SH2, // HB 系列录像机
	NET_DVR_MPEG4_GBE, // GBE 系列录像机
	NET_DVR_MPEG4_NVSII, // II 代网络视频服务器
	NET_DVR_STD_NEW, // 新标准配置协议
	NET_DVR_DDNS, // DDNS 服务器
	NET_DVR_ATM, // ATM 机
	NET_NB_SERIAL, // 二代非实时 NB 系列机器
	NET_LN_SERIAL, // LN 系列产品
	NET_BAV_SERIAL, // BAV 系列产品
	NET_SDIP_SERIAL, // SDIP 系列产品
	NET_IPC_SERIAL, // IPC 系列产品
	NET_NVS_B, // NVS B 系列
	NET_NVS_C, // NVS H 系列
	NET_NVS_S, // NVS S 系列
	NET_NVS_E, // NVS E 系列
	NET_DVR_NEW_PROTOCOL,
	// 从 QueryDevState 中查询设备类型,以字符串格式
	NET_NVD_SERIAL, // 解码器
	NET_DVR_N5, // N5
	// 省略其它枚举，详细介绍请参见 dhnetsdk.h
	NET_DVR_N52, // N52
	NET_DVR_N56, // N56
	NET_ESS_SERIAL, // ESS
	NET_IVS_PC, // 人数统计服务器
	NET_PC_NVR, // pc-nvr
	NET_DSCON, // 大屏控制器
	NET_EVS, // 网络视频存储服务器
	NET_EIVS, // 嵌入式智能分析视频系统
	NET_DVR_N6, // DVR-N6
	NET_UDS, // 万能解码器
	NET_AF6016, // 银行报警主机
	NET_AS5008, // 视频网络报警主机
	NET_AH2008, // 网络报警主机
	NET_A_SERIAL, // 报警主机系列
	NET_BSC_SERIAL, // 门禁系列产品
	NET_NVS_SERIAL, // NVS 系列产品
	NET_VTO_SERIAL, // VTO 系列产品
	NET_VTNC_SERIAL, // VTNC 系列产品
	NET_TPC_SERIAL, // TPC 系列产品，即热成像设备
	}NET_DEVICE_TYPE ;

5.2 优化获取硬盘信息接口枚举 EM_OPTIMIZE_TYPE

表5-2 EM_OPTIMIZE_TYPE

选项	说明
枚举描述	设置内部优化方案，用于选择不同的优化方式
枚举定义	typedef enum tagEmOptimizeType { EM_OPT_TYPE_DEFAULT = 0, /// 默认不优化 EM_OPT_TYPE_MOBILE_V1 = 1, /// 和选项 EM_OPT_TYPE_MOBILE_OPTION 冲突，两者只能使用一个，不支持异步登陆优化，手机使用优化,开启后，默认过滤设备属性和硬盘信息. EM_OPT_TYPE_P2P_NETPARAM_V1 = 2, /// P2P 网络参数设置，pParam 为 NET_PARAM* EM_OPT_TYPE_MOBILE_OPTION = 3, /// 和选项 EM_OPT_TYPE_MOBILE_V1 冲突，两者只能使用一个,支持异步登陆优化，手机优化选项, 参数 int, 按位从低到高, 传入的值参考枚举类型 EM_OPTTYPE_MOBILE_TYPE. EM_OPT_TYPE_LOGIN_GET_ENCRYPT = 4, /// 登录时获取秘钥信息，减少拉流耗时，pParam 为 int*， 1 表示开启优化 }EM_OPTIMIZE_TYPE;

5.3 登录方式类型枚举 EM_LOGIN_SPAC_CAP_TYPE

表5-3 EM_LOGIN_SPAC_CAP_TYPE

选项	说明
枚举描述	登录方式类型枚举，用于选择不同的登录方式

选项	说明

枚举定义	typedef enum tagEM_LOGIN_SPAC_CAP_TYPE
	{
	EM_LOGIN_SPEC_CAP_TCP= 0, // TCP 登录，默认方式
	EM_LOGIN_SPEC_CAP_ANY = 1, // 无条件登录
	EM_LOGIN_SPEC_CAP_SERVER_CONN = 2, // 主动注册的登录
	EM_LOGIN_SPEC_CAP_MULTICAST = 3, // 组播登录，默认方式
	EM_LOGIN_SPEC_CAP_UDP= 4, // UDP 方式下的登录
	EM_LOGIN_SPEC_CAP_MAIN_CONN_ONLY= 6,
	// 只建主连接下的登录
	EM_LOGIN_SPEC_CAP_SSL= 7, // SSL 加密方式登录
	EM_LOGIN_SPEC_CAP_INTELLIGENT_BOX= 9,
	// 登录智能盒远程设备
	EM_LOGIN_SPEC_CAP_NO_CONFIG= 10,
	// 登录设备后不做取配置操作
	EM_LOGIN_SPEC_CAP_U_LOGIN= 11, // 用 U 盾设备的登录
	EM_LOGIN_SPEC_CAP_LDAP= 12, // LDAP 方式登录
	EM_LOGIN_SPEC_CAP_AD= 13, // AD（ActiveDirectory）登录方式
	EM_LOGIN_SPEC_CAP_RADIUS = 14, // Radius 登录方式
	EM_LOGIN_SPEC_CAP_SOCKET_5 = 15, // Socks5 登录方式
	EM_LOGIN_SPEC_CAP_CLOUD= 16, // 云登录方式
	EM_LOGIN_SPEC_CAP_AUTH_TWICE= 17, // 二次鉴权登录方式
	EM_LOGIN_SPEC_CAP_TS = 18, // TS 码流客户端登录方式
	EM_LOGIN_SPEC_CAP_P2P = 19, // 为 P2P 登录方式
	EM_LOGIN_SPEC_CAP_MOBILE= 20, // 手机客户端登录
	EM_LOGIN_SPEC_CAP_INVALID// 无效的登录方式
	}EM_LOGIN_SPAC_CAP_TYPE;

5.4 预览类型枚举 DH_RealPlayType

表5-4 DH_RealPlayType

选项	说明
枚举描述	预览类型枚举，对应 CLIENT_RealPlayEx

选项	说明

枚举定义	typedef enum _RealPlayType
	{
	DH_RType_Realplay = 0, // 实时预览
	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
	} DH_RealPlayType;

5.5 录像查询类型枚举 EM_QUERY_RECORD_TYPE

表5-5 EM_QUERY_RECORD_TYPE

选项	说明
枚举描述	录像查询类型枚举

枚举定义	typedef enum tagEmQueryRecordType
	{
	EM_RECORD_TYPE_ALL = 0, // 所有录像
	EM_RECORD_TYPE_ALARM = 1, // 外部报警录像
	EM_RECORD_TYPE_MOTION_DETECT = 2, // 动态检测报警录像
	EM_RECORD_TYPE_ALARM_ALL = 3, // 所有报警录像
	EM_RECORD_TYPE_CARD = 4, // 卡号查询
	EM_RECORD_TYPE_CONDITION = 5, // 按条件查询
	EM_RECORD_TYPE_JOIN = 6, // 组合查询
	EM_RECORD_TYPE_CARD_PICTURE = 8, // 按卡号查询图片，HB-U、NVS 等使用
	EM_RECORD_TYPE_PICTURE = 9, // 查询图片，HB-U、NVS 等使用
	EM_RECORD_TYPE_FIELD = 10, // 按字段查询
	EM_RECORD_TYPE_INTELLI_VIDEO = 11, // 智能录像查询
	EM_RECORD_TYPE_TRANS_DATA = 16, // 查询透明串口数据录像
	EM_RECORD_TYPE_IMPORTANT = 17, // 查询重要录像
	EM_RECORD_TYPE_TALK_DATA = 18, // 查询录音文件
	EM_RECORD_TYPE_INVALID = 256, // 无效的查询类型
	}EM_QUERY_RECORD_TYPE;

5.6 设备工作模式类型枚举 EM_USEDEV_MODE

表5-6 EM_USEDEV_MODE

选项	说明
枚举描述	设备工作模式类型枚举（部分模式不在本次文档描述范围内，故对应的扩展数据类型未做说明）
枚举定义	typedef enum __EM_USEDEV_MODE { DH_TALK_CLIENT_MODE,// 设置客户端方式进行语音对讲(扩展数据为 NULL) DH_TALK_SERVER_MODE,// 设置服务器方式进行语音对讲(扩展数据为 NULL) DH_TALK_ENCODE_TYPE,// 设置语音对讲编码格式(扩展数据为 DHDEV_TALKDECODE_INFO*) DH_ALARM_LISTEN_MODE,// 设置报警订阅方式(扩展数据为 NULL) DH_CONFIG_AUTHORITY_MODE,// 设置通过权限进行配置管理(扩展数据为 NULL) DH_TALK_TALK_CHANNEL,// 设置对讲通道(扩展数据为 int*，指针指向地址取值范围 0~MaxChannel-1) DH_RECORD_STREAM_TYPE,// 设置待查询及按时间回放的录像码流类型(扩展数据为 int*，指针指向地址取值 0-主辅码流,1-主码流,2-辅码流) DH_TALK_SPEAK_PARAM, // 设置语音对讲喊话参数（扩展数据为 NET_SPEAK_PARAM*） DH_RECORD_TYPE, // 设置按时间录像回放及下载的录像文件类型(详见 NET_RECORD_TYPE) DH_TALK_MODE3, // 设置三代设备的语音对讲参数（扩展数据为 NET_TALK_EX） DH_PLAYBACK_REALTIME_MODE, // 设置实时回放功能（扩展数据为 int*，指针指向地址取值 0-关闭，1 开启） DH_TALK_TRANSFER_MODE, // 设置语音对讲是否为转发模式(扩展数据为 NET_TALK_TRANSFER_PARAM*) DH_TALK_VT_PARAM, //设置 VT 对讲参数，对应结构体 NET_VT_TALK_PARAM DH_TARGET_DEV_ID,//设置目标设备标示符，用以查询新系统能力(非 0-转发系统能力消息) } EM_USEDEV_MODE;

5.7 焦距模式类型枚举 EM_SUPPORT_FOCUS_MODE

表5-7 EM_SUPPORT_FOCUS_MODE

选项	说明
枚举描述	支持的焦距模式类型枚举
枚举定义	typedef enum tagSUPPORT_FOCUS_MODE { ENUM_SUPPORT_FOCUS_CAR= 1,// 看清车模式 ENUM_SUPPORT_FOCUS_PLATE= 2,// 看清车牌模式 ENUM_SUPPORT_FOCUS_PEOPLE= 3,// 看清人模式 ENUM_SUPPORT_FOCUS_FACE= 4,// 看清人脸模式 }EM_SUPPORT_FOCUS_MODE;

5.8 通用云台控制命令枚举 DH_PTZ_ControlType

表5-8 DH_PTZ_ControlType

选项	说明
枚举描述	通用云台控制命令枚举

枚举定义	typedef enum _PTZ_ControlType
	{
	DH_PTZ_UP_CONTROL = 0, // 上，lParam2：垂直/水平移动速度，有效范围(1-8)
	DH_PTZ_DOWN_CONTROL, // 下，lParam2：垂直/水平移动速度，有效范围(1-8)
	DH_PTZ_LEFT_CONTROL, // 左，lParam2：垂直/水平移动速度，有效范围(1-8)
	DH_PTZ_RIGHT_CONTROL, // 右，lParam2：垂直/水平移动速度，有效范围(1-8)
	DH_PTZ_ZOOM_ADD_CONTROL, // 变倍+，lParam2：速度，有效范围(1-8)
	DH_PTZ_ZOOM_DEC_CONTROL, // 变倍-，lParam2：速度，有效范围(1-8)
	DH_PTZ_FOCUS_ADD_CONTROL, // 调焦+，lParam2：速度，有效范围(1-8)
	DH_PTZ_FOCUS_DEC_CONTROL, // 调焦-，lParam2：速度，有效范围(1-8)
	DH_PTZ_APERTURE_ADD_CONTROL, // 光圈+，lParam2：速度，有效范围(1-8)
	DH_PTZ_APERTURE_DEC_CONTROL, // 光圈-，lParam2：速度，有效范围(1-8)
	DH_PTZ_POINT_MOVE_CONTROL, // 转至预置点，lParam2：表示预置点号
	DH_PTZ_POINT_SET_CONTROL, // 设置，lParam2：表示预置点号
	DH_PTZ_POINT_DEL_CONTROL, // 删除，lParam2：表示预置点号
	DH_PTZ_POINT_LOOP_CONTROL, // 点间巡航，lParam1：表示巡航路线，lParam3：76 开始；96 停止
	DH_PTZ_LAMP_CONTROL// 灯光雨刷，lParam1：表示开关控制，1:开启，0:关闭
	} DH_PTZ_ControlType;

5.9 云台控制扩展命令枚举 DH_EXTPTZ_ControlType

表5-9 DH_EXTPTZ_ControlType

选项	说明
枚举描述	云台控制扩展命令枚举

选项	说明

枚举定义	typedef enum _EXTPTZ_ControlType
	{
	DH_EXTPTZ_LEFTTOP = 0x20,// 左上
	DH_EXTPTZ_RIGHTTOP,// 右上
	DH_EXTPTZ_LEFTDOWN,// 左下
	DH_EXTPTZ_RIGHTDOWN,// 右下
	DH_EXTPTZ_ADDTOLOOP,// 加入预置点到巡航，lParam1：表示巡航路线；lParam2 表示预置点号
	DH_EXTPTZ_DELFROMLOOP,// 删除巡航中预置点，lParam1：表示巡航路线；lParam2 表示预置点号
	DH_EXTPTZ_CLOSELOOP,// 清除巡航 lParam1：表示巡航路线
	DH_EXTPTZ_STARTPANCRUISE,// 开始水平旋转
	DH_EXTPTZ_STOPPANCRUISE,// 停止水平旋转
	DH_EXTPTZ_SETLEFTBORDER,// 设置左边界
	DH_EXTPTZ_SETRIGHTBORDER,// 设置右边界
	DH_EXTPTZ_STARTLINESCAN,// 开始线扫
	DH_EXTPTZ_CLOSELINESCAN,// 停止线扫
	DH_EXTPTZ_SETMODESTART,// 设置模式开始 模式线路
	DH_EXTPTZ_SETMODESTOP,// 设置模式结束 模式线路
	DH_EXTPTZ_RUNMODE,// 运行模式 模式线路
	DH_EXTPTZ_STOPMODE,// 停止模式 模式线路
	DH_EXTPTZ_DELETEMODE,// 清除模式 模式线路
	DH_EXTPTZ_REVERSECOMM,// 翻转命令
	// 省略其它枚举，详细介绍请参见 dhnetsdk.h
	DH_EXTPTZ_HORSECTORSCAN = 0x4B, // 水平扇扫(param4 对应 PTZ_CONTROL_SECTORSCAN,param1、param2、param3 无效)
	DH_EXTPTZ_VERSECTORSCAN = 0x4C, // 垂直扇扫(param4 对应 PTZ_CONTROL_SECTORSCAN,param1、param2、param3 无效)
	DH_EXTPTZ_SET_ABS_ZOOMFOCUS = 0x4D, // 设定绝对焦距、聚焦值,param1 为焦距，范围：[0,255]，param2 为聚焦，范围：[0,255]，param3、param4 无效。
	DH_EXTPTZ_SET_FISHEYE_EPTZ = 0x4E, // 控制鱼眼电子云台，param4 对应结构 PTZ_CONTROL_SET_FISHEYE_EPTZ
	DH_EXTPTZ_UP_TELE = 0x70,// 上 + TELE param1=速度(1-8)，下同
	DH_EXTPTZ_DOWN_TELE,// 下 + TELE
	DH_EXTPTZ_LEFT_TELE,// 左 + TELE
	DH_EXTPTZ_RIGHT_TELE,// 右 + TELE
	DH_EXTPTZ_LEFTUP_TELE,// 左上 + TELE
	DH_EXTPTZ_LEFTDOWN_TELE, // 左下 + TELE
	DH_EXTPTZ_TIGHTUP_TELE,// 右上 + TELE
	DH_EXTPTZ_RIGHTDOWN_TELE, // 右下 + TELE
	DH_EXTPTZ_UP_WIDE, // 上 + WIDE param1=速度(1-8)，下同
	DH_EXTPTZ_DOWN_WIDE,// 下 + WIDE
	DH_EXTPTZ_LEFT_WIDE,// 左 + WIDE
	DH_EXTPTZ_RIGHT_WIDE,// 右 + WIDE
	DH_EXTPTZ_LEFTUP_WIDE,// 左上 + WIDE
	DH_EXTPTZ_LEFTDOWN_WIDE, // 左下 + WIDE
	DH_EXTPTZ_TIGHTUP_WIDE, // 右上 + WIDE
	DH_EXTPTZ_RIGHTDOWN_WIDE, // 右下 + WIDE
	DH_EXTPTZ_TOTAL,// 最大命令值
	} DH_EXTPTZ_ControlType;

5.10 语音编码类型枚举 DH_TALK_CODING_TYPE

表5-10 DH_TALK_CODING_TYPE

选项	说明
枚举描述	语音编码类型枚举
枚举定义	typedef enum __TALK_CODING_TYPE { DH_TALK_DEFAULT = 0,// 无头 PCM DH_TALK_PCM = 1,// 带头 PCM DH_TALK_G711a,// G711a DH_TALK_AMR, // AMR DH_TALK_G711u,// G711u DH_TALK_G726, // G726 DH_TALK_G723_53,// G723_53 DH_TALK_G723_63,// G723_63 DH_TALK_AAC,// AAC DH_TALK_OGG,// OGG DH_TALK_G729 = 10,// G729 DH_TALK_MPEG2,// MPEG2 DH_TALK_MPEG2_Layer2,// MPEG2-Layer2 DH_TALK_G722_1,// G.722.1 DH_TALK_ADPCM= 21,// ADPCM DH_TALK_MP3 = 22,// MP3 } DH_TALK_CODING_TYPE;

5.11 设备控制类型枚举 CtrlType

表5-11 CtrlType

选项	说明
枚举描述	设备控制类型枚举，对应 CLIENT_ControlDeviceEx 接口

选项	说明

枚举定义	typedef enum _CtrlType
	{
	DH_CTRL_REBOOT = 0, // 重启设备
	DH_CTRL_SHUTDOWN, // 关闭设备
	DH_CTRL_DISK, // 硬盘管理
	// 省略其它枚举，详细介绍请参见 dhnetsdk.h
	NET_CTRL_OPEN_DOOR_CONTINUE)
	// 以下命令只在 CLIENT_ControlDeviceEx 上有效
	DH_CTRL_THERMO_GRAPHY_ENSHUTTER = 0x10000, // 设置热成像快门启用/禁用, pInBuf= NET_IN_THERMO_EN_SHUTTER*, pOutBuf= NET_OUT_THERMO_EN_SHUTTER *
	DH_CTRL_RADIOMETRY_SETOSDMARK, // 设置测温项的 osd 为高亮, pInBuf= NET_IN_RADIOMETRY_SETOSDMARK*, pOutBuf= NET_OUT_RADIOMETRY_SETOSDMARK *
	DH_CTRL_AUDIO_REC_START_NAME, // 开启音频录音并得到录音名, pInBuf = NET_IN_AUDIO_REC_MNG_NAME *, pOutBuf = NET_OUT_AUDIO_REC_MNG_NAME *
	DH_CTRL_AUDIO_REC_STOP_NAME, // 关闭音频录音并返回文件名称, pInBuf = NET_IN_AUDIO_REC_MNG_NAME *, pOutBuf = NET_OUT_AUDIO_REC_MNG_NAME *
	DH_CTRL_SNAP_MNG_SNAP_SHOT, // 即时抓图(又名手动抓图), pInBuf = NET_IN_SNAP_MNG_SHOT *, pOutBuf = NET_OUT_SNAP_MNG_SHOT *
	DH_CTRL_LOG_STOP, // 强制同步缓存数据到数据库并关闭数据库, pInBuf = NET_IN_LOG_MNG_CTRL *, pOutBuf = NET_OUT_LOG_MNG_CTRL *
	DH_CTRL_LOG_RESUME, // 恢复数据库, pInBuf = NET_IN_LOG_MNG_CTRL *, pOutBuf = NET_OUT_LOG_MNG_CTRL *
	DH_CTRL_POS_ADD, // 增加一个 Pos 设备, pInBuf = NET_IN_POS_ADD *, pOutBuf = NET_OUT_POS_ADD *
	DH_CTRL_POS_REMOVE, // 删除一个 Pos 设备, pInBuf = NET_IN_POS_REMOVE *, pOutBuf = NET_OUT_POS_REMOVE *
	DH_CTRL_POS_REMOVE_MULTI, // 批量删除 Pos 设备, pInBuf = NET_IN_POS_REMOVE_MULTI *, pOutBuf = NET_OUT_POS_REMOVE_MULTI *
	DH_CTRL_POS_MODIFY, // 修改一个 Pos 设备, pInBuf = NET_IN_POS_ADD *, pOutBuf = NET_OUT_POS_ADD *
	DH_CTRL_SET_SOUND_ALARM, // 触发有声报警, pInBuf = NET_IN_SOUND_ALARM *, pOutBuf = NET_OUT_SOUND_ALARM *
	DH_CTRL_AUDIO_MATRIX_SILENCE, // 音频举证一键静音控制(对应 pInBuf = NET_IN_AUDIO_MATRIX_SILENCE, pOutBuf = NET_OUT_AUDIO_MATRIX_SILENCE)
	DH_CTRL_MANUAL_UPLOAD_PICTURE, // 设置手动上传, pInBuf = NET_IN_MANUAL_UPLOAD_PICTURE *, pOutBUf = NET_OUT_MANUAL_UPLOAD_PICTURE *
	DH_CTRL_REBOOT_NET_DECODING_DEV, // 重启网络解码设备, pInBuf = NET_IN_REBOOT_NET_DECODING_DEV *, pOutBuf = NET_OUT_REBOOT_NET_DECODING_DEV *
	} CtrlType;

5.12 视频压缩格式类型枚举 CFG_VIDEO_COMPRESSION

表5-12 CFG_VIDEO_COMPRESSION

选项	说明
枚举描述	视频压缩格式类型枚举
枚举定义	typedef enum tagCFG_VIDEO_COMPRESSION { VIDEO_FORMAT_MPEG4, // MPEG4 VIDEO_FORMAT_MS_MPEG4, // MS-MPEG4 VIDEO_FORMAT_MPEG2, // MPEG2 VIDEO_FORMAT_MPEG1, // MPEG1 VIDEO_FORMAT_H263, // H.263 VIDEO_FORMAT_MJPG, // MJPG VIDEO_FORMAT_FCC_MPEG4, // FCC-MPEG4 VIDEO_FORMAT_H264, // H.264 VIDEO_FORMAT_H265, // H.265 } CFG_VIDEO_COMPRESSION;

5.13 码流控制模式枚举 CFG_BITRATE_CONTROL

表5-13 CFG_BITRATE_CONTROL

选项	说明
枚举描述	码流控制模式枚举
枚举定义	typedef enum tagCFG_BITRATE_CONTROL { BITRATE_CBR, // 固定码流 BITRATE_VBR, // 可变码流 } CFG_BITRATE_CONTROL;

5.14 画质类型枚举 CFG_IMAGE_QUALITY

表5-14 CFG_IMAGE_QUALITY

选项	说明
枚举描述	画质类型枚举

枚举定义	typedef enum tagCFG_IMAGE_QUALITY
	{
	IMAGE_QUALITY_Q10 = 1, // 图像质量 10%
	IMAGE_QUALITY_Q30, // 图像质量 30%
	IMAGE_QUALITY_Q50, // 图像质量 50%
	IMAGE_QUALITY_Q60, // 图像质量 60%
	IMAGE_QUALITY_Q80, // 图像质量 80%
	IMAGE_QUALITY_Q100, // 图像质量 100%
	} CFG_IMAGE_QUALITY;

5.15 H264 编码级别枚举 CFG_H264_PROFILE_RANK

表5-15 CFG_H264_PROFILE_RANK

选项	说明
枚举描述	H264 编码级别枚举

枚举定义	typedef enum tagCFG_H264_PROFILE_RANK
	{
	PROFILE_BASELINE = 1, // 提供 I/P 帧，仅支持 progressive(逐行扫描)和 CAVLC
	PROFILE_MAIN, // 提供 I/P/B 帧，支持 progressiv 和 interlaced，提供 CAVLC 或 CABAC
	PROFILE_EXTENDED, // 提供 I/P/B/SP/SI 帧，仅支持 progressive(逐行扫描)和 CAVLC
	PROFILE_HIGH, // 即 FRExt，Main_Profile 基础上新增：8x8 intra prediction(8x8 帧内预测), custom quant(自定义量化), lossless video coding(无损视频编码), 更多的 yuv 格式
	}CFG_H264_PROFILE_RANK;

5.16 音频编码模式枚举 CFG_AUDIO_FORMAT

表5-16 CFG_AUDIO_FORMAT

选项	说明
枚举描述	音频编码模式枚举
枚举定义	typedef enum tatCFG_AUDIO_FORAMT { AUDIO_FORMAT_G711A, // G711a AUDIO_FORMAT_PCM, // PCM AUDIO_FORMAT_G711U, // G711u AUDIO_FORMAT_AMR, // AMR AUDIO_FORMAT_AAC, // AAC } CFG_AUDIO_FORMAT;

5.17 搜索类型枚举 EM_SEND_SEARCH_TYPE

表5-17 EM_SEND_SEARCH_TYPE

选项	说明
枚举描述	下发搜索类型枚举
枚举定义	typedef enum tagEM_SEND_SEARCH_TYPE { EM_SEND_SEARCH_TYPE_MULTICAST_AND_BROADCAST, // 组播和广播搜索 EM_SEND_SEARCH_TYPE_MULTICAST, // 组播搜索 EM_SEND_SEARCH_TYPE_BROADCAST, // 广播搜索 }EM_SEND_SEARCH_TYPE;

5.18 视频预览断开事件类型 EM_REALPLAY_DISCONNECT_EVENT_TYPE

表5-18 EM_REALPLAY_DISCONNECT_EVENT_TYPE

选项	说明
枚举描述	视频预览断开事件类型
枚举定义	typedef enum _EM_REALPLAY_DISCONNECT_EVENT_TYPE { DISCONNECT_EVENT_REAVE, // 表示高级用户抢占低级用户资源 DISCONNECT_EVENT_NETFORBID, // 禁止入网 DISCONNECT_EVENT_SUBCONNECT, // 动态子链接断开 }EM_REALPLAY_DISCONNECT_EVENT_TYPE;

5.19 升级安装执行操作 EM_NET_UPGRADE_INSTALL_TYPE

表5-19 EM_NET_UPGRADE_INSTALL_TYPE

选项	说明
枚举描述	升级安装执行操作
枚举定义	typedef enum tag_EM_NET_UPGRADE_INSTALL_TYPE { EM_UPGRADER_INSTALL_PREPAREEX, ///升级准备 EM_UPGRADER_INSTALL_APPEND_DATA, ///升级数据 EM_UPGRADER_INSTALL_EXECUTE, ///执行升级 EM_UPGRADER_INSTALL_GETSTATE, ///获取升级状态 EM_UPGRADER_INSTALL_CANCEL, ///取消升级 EM_UPGRADER_INSTALL_FIRMWAREEX, ///指定升级包地址进行升级 }EM_NET_UPGRADE_INSTALL_TYPE;

5.20 APP 的运行状态 EM_APP_RUNNING_STATE

表5-20 EM_APP_RUNNING_STATE

选项	说明
枚举描述	APP 的运行状态
枚举定义	typedef enum tagEM_APP_RUNNING_STATE { EM_APP_RUNNING_STATE_UNKNOWN, /// 未知 EM_APP_RUNNING_STATE_RUNNING, /// 运行中 EM_APP_RUNNING_STATE_STOP, /// 已停止 EM_APP_RUNNING_STATE_ERROR, /// 异常状态 } EM_APP_RUNNING_STATE;

5.21 APP 的证书状态 EM_APP_LICENSE_STATE

表5-21 EM_APP_LICENSE_STATE

选项	说明
枚举描述	APP 的证书状态
枚举定义	typedef enum tagEM_APP_LICENSE_STATE { EM_APP_LICENSE_STATE_UNKNOEN, /// 未知 EM_APP_LICENSE_STATE_IN_TRAL, /// 证书试用期 EM_APP_LICENSE_STATE_IN_LICENSE, /// 证书正式合法 EM_APP_LICENSE_STATE_EXPIRED, /// 证书过期 }EM_APP_LICENSE_STATE;

5.22 APP 的调试状态 EM_APP_DEBUG_STATE

表5-22 EM_APP_DEBUG_STATE

选项	说明
枚举描述	APP 的调试状态
枚举定义	typedef enum tagEM_APP_DEBUG_STATE { EM_APP_DEBUG_STATE_UNKNOWN, /// 未知 EM_APP_DEBUG_STATE_ENABLE, /// 开启调试 EM_APP_DEBUG_STATE_DISABLE, /// 关闭调试 }EM_APP_DEBUG_STATE;

5.23 升级安装准备操作 EM_NET_NEXT_OPERATE

表5-23 EM_NET_NEXT_OPERATE

选项	说明
枚举描述	升级安装准备操作
枚举定义	typedef enum tagEM_NET_NEXT_OPERATE { EM_NET_NEXT_OPERATE_UNKNOWN = 0, ///未知操作 EM_NET_NEXT_OPERATE_INSTALL, ///安装 App; EM_NET_NEXT_OPERATE_UPDATE, /// 更新 App; }EM_NET_NEXT_OPERATE;

5.24 升级安装执行操作 EM_NET_UPGRADE_INSTALL_TYPE

表5-24 EM_NET_UPGRADE_INSTALL_TYPE

选项	说明
枚举描述	升级安装执行操作
枚举定义	typedef enum tag_EM_NET_UPGRADE_INSTALL_TYPE { EM_UPGRADER_INSTALL_PREPAREEX, ///升级准备 EM_UPGRADER_INSTALL_APPEND_DATA, ///升级数据 EM_UPGRADER_INSTALL_EXECUTE, ///执行升级 EM_UPGRADER_INSTALL_GETSTATE, ///获取升级状态 EM_UPGRADER_INSTALL_CANCEL, ///取消升级 EM_UPGRADER_INSTALL_FIRMWAREEX, ///指定升级包地址进行升级 }EM_NET_UPGRADE_INSTALL_TYPE;

第 6 章 接口函数定义

6.1 SDK 初始化接口 CLIENT_Init

表6-1 CLIENT_Init

选项	说明
接口描述	SDK 初始化接口，在应用程序初始化时调用。
前置条件	无
函数	BOOL CLIENT_Init(fDisConnect cbDisConnect, LDWORD dwUser);
参数	● cbDisConnect *[in]* 断线回调函数，当原本在线设备出现断线时，SDK 通过该回调函数通知用户，回调的信息包括登录 ID，设备 IP，登录端口等信息，详细请参见“3.1 断线回调函数 fDisConnect”。当回调函数设置为 0 时，表示禁止回调。 ● dwUser *[in]* 用户数据，当断线回调函数不为 0 时，SDK 通过断线回调函数 fDisConnect 将该数据返回给用户，以便用户后续操作。
返回值	成功返回 TRUE，失败返回 FALSE。

选项	说明

使用示例	// 不建议在 SDK 的回调函数中调用 SDK 接口，除非是调用 CLIENT_GetLastError 来获取当前线程的错误码
	// 设备断线回调函数
	// 当设备出现断线时，SDK 会调用该函数，在 CLIENT_Init 中设置该回调函数
	void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser)
	{
	printf("Call DisConnectFunc\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
	printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
	}
	**********上面为回调函数定义，下面为接口使用示例**********
	// 初始化 SDK
	g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
	if (FALSE == g_bNetSDKInitFlag)
	{
	printf("Initialize client SDK failed; \n");
	return;
	}
	else
	{
	printf("Initialize client SDK done; \n");
	}

注释	在调用其他 SDK 接口前，必须先调用此接口。 若重复调用该接口，则以第一次为准。

6.2 SDK 清理接口 CLIENT_Cleanup

表6-2 CLIENT_Cleanup

选项	说明
接口描述	SDK 清理接口
前置条件	已调用 CLIENT_Init 初始化接口。
函数	void CLIENT_Cleanup( );
参数	无
返回值	无

选项	说明

使用示例	// 清理初始化资源
	printf("CLIENT_Cleanup!\n");
	CLIENT_Cleanup();

注释	在应用程序关闭时调用，释放占用的资源，在所有的 SDK 函数之后调用。

6.3 获取 SDK 的版本信息接口 CLIENT_GetSDKVersion

表6-3 CLIENT_GetSDKVersion

选项	说明
接口描述	获取 SDK 的版本信息接口
前置条件	已调用 CLIENT_Init 初始化接口。
函数	DWORD CLIENT_GetSDKVersion( );
参数	无
返回值	返回值为版本号，类似 34219000，即 3.42 19000 版本号

使用示例	// 获取 SDK 版本信息
	DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
	printf("NetSDK version is [%d]\n", dwNetSdkVersion);

注释	无

6.4 获取错误码接口 CLIENT_GetLastError

表6-4 CLIENT_GetLastError

选项	说明
接口描述	获取错误码接口，获取当前线程的错误码
前置条件	已调用 CLIENT_Init 初始化接口。
函数	DWORD CLIENT_GetLastError(void);
参数	无
返回值	当前线程对应的错误码

选项	说明

使用示例	方案一：
	按照十六进制打印错误码，在 dhnetsdk.h 搜索十六进制数值找到相应的解释
	// 例如：
	printf("Last Error[%x]\n" , CLIENT_GetLastError());
	输出打印错误码为 0x80000017，头文件 dhnetsdk.h 搜索对应 NET_NOT_SUPPORTED
	方案二：
	// 根据错误码，可以在 dhnetsdk.h 中找到相应的解释，此处打印的是 16 进制，头文件中是十进制，其中的转换需注意
	// 例如：
	// #define NET_NOT_SUPPORTED_EC(23)
	// 当前 SDK 未支持该功能，对应的错误码为 0x80000017, 23 对应的 16 进制为 0x17
	printf("Last Error[%x]\n" , CLIENT_GetLastError());

注释	当线程 SDK 接口调用失败后，调用该接口 由于错误码较多，这里不一一举例，用户可在 dhnetsdk.h 中搜索如下字段： // 错误类型代号,对应 CLIENT_GetLastError 接口的返回值 #define _EC(x) (0x80000000|x) 即可找到相应错误码的说明

6.5 设置断线重连成功回调函数接口

CLIENT_SetAutoReconnect

表6-5 CLIENT_SetAutoReconnect

选项	说明
接口描述	设置断线重连成功回调函数接口，设置后 SDK 内部检测到设备断线则进行自动重连。
前置条件	已调用 CLIENT_Init 初始化接口。
函数	void CLIENT_SetAutoReconnect(HaveReConnect cbAutoConnect, DWORD dwUser);
参数	● cbAutoConnect *[in]* 断线重连成功回调函数，SDK 在断线重连成功后回调该接口提示用户重连成功。 ● dwUser *[in]* 用户数据，由用户传入，在断线重连成功回调函数中返回供用户使用。
返回值	无

选项	说明

使用示例	// 不建议在 SDK 的回调函数中调用 SDK 接口，除非是调用 CLIENT_GetLastError 来获取当前线程的错误码。
	// 断线重连成功回调函数。
	// 当已断线的设备重连成功时，SDK 会调用该函数，在 CLIENT_SetAutoReconnect 中设置该回调函数。
	void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
	{
	printf("Call HaveReConnect\n");
	printf("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
	printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
	}
	**********上面为回调函数定义，下面为接口使用示例**********
	// 设置断线重连回调接口，设置过断线重连成功回调函数后，当设备出现断线情况，SDK 内部会自动进行重连操作。
	CLIENT_SetAutoReconnect(&HaveReConnect, 0);

注释	当调用该接口设置过断线重连成功回调函数后，设备端发生断线情况，SDK 会不断重连设备，如果重连成功，则通过断线重连成功回调函数通知客户。若没有调用过该接口或者调用该接口但是断线重连成功回调函数为空时，设备端发生断线，SDK 不会尝试重连设备。

6.6 设置连接设备超时时间和尝试次数接口 CLIENT_SetConnectTime

表6-6 CLIENT_SetConnectTime

选项	说明
接口描述	设置连接设备超时时间和尝试次数接口。
前置条件	已调用 CLIENT_Init 初始化接口。
函数	void CLIENT_SetConnectTime(int nWaitTime, int nTryTimes);
参数	● nWaitTime *[in]* 每次登录时，等待设备响应超时时间。 ● nTryTimes *[in]* 每次登录时，连接尝试登录设备次数。
返回值	无

选项	说明

使用示例	// 设置连接设备超时时间和尝试次数
	// 此操作为可选操作
	int nWaitTime = 5000; // 超时时间设置为 5 s
	int nTryTimes = 3; // 若出现超时，尝试登录 3 次
	CLIENT_SetConnectTime(nWaitTime, nTryTimes);

注释	若不调用 CLIENT_SetConnectTime 接口，则等待设备响应超时时间默认为 5s，连接尝试登录设备次数默认为 1 次。

6.7 设置登录网络环境接口 CLIENT_SetNetworkParam

表6-7 CLIENT_SetNetworkParam

选项	说明
接口描述	设置登录网络环境接口
前置条件	已调用 CLIENT_Init 初始化接口
函数	void CLIENT_SetNetworkParam(NET_PARAM *pNetParam);
参数	pNetParam *[in]* 用于提供网络设置的参数，具体请参见章节 4.2 设置登录相关参数结构体 NET_PARAM。
返回值	无

使用示例	// 设置网络登录参数，参数中包含登录连接尝试次数以及超时时间
	NET_PARAM stuNetParm = {0};
	stuNetParm.nWaittime = 10000; // 将登录时超时时间改为 10 s，其他参数依然使用默认值
	CLIENT_SetNetworkParam(&stuNetParm);

注释	无

6.8 优化获取硬盘信息方案接口 CLIENT_SetOptimizeMode

表6-8 CLIENT_SetOptimizeMode

选项	说明
接口描述	优化获取硬盘信息
前置条件	已调用 CLIENT_Init 初始化接口
函数	BOOL CLIENT_SetOptimizeMode(EM_OPTIMIZE_TYPE emType, void *pParam);
参数	● pstInParam *[in]* 入参，emType 方案类型，具体请参见 EM_OPTIMIZE_TYPE 枚举定义。 *[in]* 入参，pParam 内存由用户申请释放，大小参照 emType 对应的结构体
返回值	成功返回 TRUE，失败返回 FALSE

选项	说明
使用示例	int opt = OPTTYPE_MOBILE_DISK_INFO; CLIENT_SetOptimizeMode(EM_OPT_TYPE_MOBILE_OPTION, &opt);
注释	默认不优化

6.9 高安全级别登录接口 CLIENT_LoginWithHighLevelSecurity

表6-9 CLIENT_LoginWithHighLevelSecurity

选项	说明
接口描述	高安全级别登录接口，用于注册用户到设备，可定义用户支持的设备能力。
前置条件	已调用 CLIENT_Init 初始化接口
函数	LLONG CLIENT_LoginWithHighLevelSecurity (NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY* pstInParam, NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY* pstOutParam);
参数	● pstInParam *[in]* 入参，具体请参见 NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY 结构体定义。 ● pstOutParam *[out]* 出参，具体请参见 NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY 结构体定义。
返回值	成功返回设备 ID，失败返回 0 登录成功之后对设备的操作都可以通过此值（设备 ID）配合 SDK 接口实现。

使用示例	// 登录设备
	NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
	memset(&stInparam, 0, sizeof(stInparam));
	stInparam.dwSize = sizeof(stInparam);
	strncpy(stInparam.szIP, “192.168.1.108”, sizeof(stInparam.szIP) - 1);
	strncpy(stInparam.szPassword, “123456”, sizeof(stInparam.szPassword) - 1);
	strncpy(stInparam.szUserName, “admin”, sizeof(stInparam.szUserName) - 1);
	stInparam.nPort = 37777;
	stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
	
	NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
	memset(&stOutparam, 0, sizeof(stOutparam));
	stOutparam.dwSize = sizeof(stOutparam);
	LLONG lLoginID =
	CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

选项	说明
注释	在初始化后就可以调用本接口注册到指定的设备，成功后将返回设备 ID，供其他相关函数调用。 建议客户使用 emSpecCap = EM_LOGIN_SPEC_CAP_TCP 为 TCP 方式下的登录。

6.10 登出接口 CLIENT_Logout

表6-10 CLIENT_Logout

选项	说明
接口描述	登出接口
函数	BOOL CLIENT_Logout(LLONG lLoginID);
参数	lLoginID *[in]* 设备登录句柄 ID。对应 CLIENT_LoginWithHighLevelSecurity 接口的返回值。
返回值	成功返回 TRUE，失败返回 FALSE。

使用示例	printf("CLIENT_Logout!\n");
	if(!CLIENT_Logout(g_lLoginHandle))
	{
	printf("CLIENT_Logout Failed!Last Error[%x]\n" ,
	CLIENT_GetLastError());
	}
	参考设备注册中的同步登录示例代码

注释	当退出设备时，该设备对应的业务功能也会停止，比如实时预览等。

6.11 开始实时预览接口 CLIENT_RealPlayEx

表6-11 CLIENT_RealPlayEx

选项	说明
接口描述	开始实时预览扩展接口，实现向已登录设备拉取实时预览码流
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	LLONG CLIENT_RealPlayEx(LLONG lLoginID, int nChannelID, HWND hWnd, DH_RealPlayType rType = DH_RType_Realplay);

选项	说明
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● nChannelID *[in]* 实时预览通道号，通道号从 0 开始 ● hWnd *[in]* *窗口*句柄，值为 0 时对数据不解码不显示图像 ● rType *[in]* 实时预览类型。 默认为 DH_RType_Realplay，具体请参见“5.4 DH_RealPlayType 枚举定义”
返回值	失败返回 0，成功返回实时预览 ID(实时预览句柄)，将作为相关函数的参数

使用示例	typedef HWND (WINAPI *PROCGETCONSOLEWINDOW)();
	PROCGETCONSOLEWINDOW GetConsoleWindow;
	// 获取控制台窗口句柄
	HMODULE hKernel32 = GetModuleHandle("kernel32");
	GetConsoleWindow =
	(PROCGETCONSOLEWINDOW)GetProcAddress(hKernel32,"GetConsole Window");
	HWND hWnd = GetConsoleWindow();
	//开启实时预览
	int nChannelID = 0; // 预览通道号
	DH_RealPlayType emRealPlayType = DH_RType_Realplay;
	g_lRealHandle = CLIENT_RealPlayEx(g_lLoginHandle, nChannelID, hWnd, emRealPlayType);
	if (g_lRealHandle == 0)
	{
	printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

注释	多画面预览时 nChannelID NVR 设备填视频输出通道号。 根据登录时获取到的设备信息，调用本接口，就可以打开任何有效的一路实时预览，显示到指定的 hWnd 窗口，成功返回实时预览 ID，用于以下对本预览通道的控制和操作。

6.12 开启实时预览转码接口 CLIENT_RealPlayByDataType

表6-12 CLIENT_RealPlayByDataType

选项	说明
接口描述	开启实时预览转码流接口。
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口

选项	说明
函数	LLON G CLIENT_RealPlayByDataType(LLONG lLoginID, const NET_IN_REALPLAY_BY_DATA_TYPE* pstInParam, NET_OUT_REALPLAY_BY_DATA_TYPE* pstOutParam, DWORD dwWaitTime);
参数	lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值。 pstInParam *[in]* 入参，具体请参见 NET_IN_REALPLAY_BY_DATA_TYPE 结构体定义 pstOutParam *[out]* 出参，具体请参见 NET_OUT_REALPLAY_BY_DATA_TYPE 结构体定义。 dwWaitTime *[in]* 用户期望搜索时间 用户根据自己需求合理设置该参数，由于该接口为同步接口，只有等待搜索时间到达才会从接口返回。
返回值	失败返回 0，成功返回实时预览 ID(实时预览句柄)，将作为相关函数的参数

使用示例	NET_IN_REALPLAY_BY_DATA_TYPE stIn = {sizeof(stIn)};
	NET_OUT_REALPLAY_BY_DATA_TYPE stOut = {sizeof(stOut)};
	stIn.emDataType = EM_REAL_DATA_TYPE_H264;
	stIn.rType = DH_RType_Realplay;
	stIn.nChannelID = 0;
	stIn.hWnd = NULL;
	stIn.dwUser = NULL;
	stIn.cbRealDataEx2 = OnDataCallBackEx;
	
	g_lRealHandle = CLIENT_RealPlayByDataType(g_lLoginHandle, &stIn, &stOut, 5000);
	if (0 == g_lRealHandle)
	{
	printf("CLIENT_RealPlayByDataType: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

注释	无

6.13 设置实时预览数据回调接口

CLIENT_SetRealDataCallBackEx

表6-13 CLIENT_SetRealDataCallBackEx

选项	说明
接口描述	设置实时预览数据回调函数扩展接口

选项	说明
前置条件	已调用 CLIENT_Init 初始化接口 已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口 已调用 CLIENT_RealPlayEx 等拉取实时预览码流接口
函数	BOOL CLIENT_SetRealDataCallBackEx(LLONG lRealHandle, fRealDataCallBackEx cbRealData, LDWORD dwUser, DWORD dwFlag);
参数	● lRealHandle *[in]* 实时预览句柄 CLIENT_RealPlayEx 等拉取实时预览码流接口的返回值。 ● cbRealData *[in]* 实时预览数据回调函数 ◇ 当 cbRealData 为 0 时，表示不回调实时预览数据； ◇ 当 cbRealData 不为 0 时，实时预览数据通过 cbRealData 回调函数回调给用户，具体参见 fRealDataCallBackEx 回调函数说明 ● dwUser *[in]* 用户数据，SDK 通过实时预览数据回调函数 fRealDataCallBackEx 将该数据返回给用户，以便用户后续操作 ● dwFlag *[in]* 回调数据选择标志 可以选择性的回调出需要的数据，对于没设置回调的数据类型就不回调出来，不同的值对应不同的数据类型，具体如下： ◇ 0x00000001：等同原来的原始数据 ◇ 0x00000002：MPEG4/H264 标准数据 ◇ 0x00000004：YUV 数据 ◇ 0x00000008：PCM 数据 ◇ 0x00000010：原始音频数据 ◇ 0x0000001f ：以上五种数据类型
返回值	成功返回 TRUE，失败返回 FALSE

选项	说明

使用示例	// 不建议在 SDK 的回调函数中调用 SDK 接口，除非是调用 CLIENT_GetLastError 来获取当前线程的错误码
	// 实时预览数据回调函数原形--扩展
	// 当收到实时预览数据时，SDK 会调用该函数，在 CLIENT_SetRealDataCallBackEx 中设置该回调函数
	// 建议用户在此回调函数中只进行保存数据的操作，即：将相应的数据拷贝到自己的存储空间，离开回调函数后再对数据做编解码等处理
	// 不建议用户在回调函数里直接对数据进行编解码等处理
	void CALLBACK RealDataCallBackEx(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser)
	{
	if (lRealHandle == g_lRealHandle)
	{
	switch(dwDataType)
	{
	case 0:
	//原始音视频混合数据
	printf("receive real data, param: lRealHandle[%p], dwDataType[%d], pBuffer[%p], dwBufSize[%d], param[%p], dwUser[%p]\n", lRealHandle, dwDataType, pBuffer, dwBufSize, param, dwUser);
	break;
	case 1:
	//标准视频数据
	break;
	case 2:
	//yuv 数据
	break;
	case 3:
	//pcm 音频数据
	break;
	case 4:
	//原始音频数据
	break;
	default:
	break;
	}
	}
	}
	**********上面为回调函数定义，下面为接口使用示例**********
	DWORD dwFlag = 0x00000001;
	if (!CLIENT_SetRealDataCallBackEx(g_lRealHandle, &RealDataCallBackEx, NULL, dwFlag))
	{
	printf("CLIENT_SetRealDataCallBackEx: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

选项	说明
注释	增加一个回调数据类型标志 dwFlag 参数，可以选择性的回调出需要的数据，对于没设置回调的数据类型不用回调。

6.14 停止实时预览接口 CLIENT_StopRealPlayEx

表6-14 CLIENT_StopRealPlayEx

选项	说明
接口描述	停止实时预览扩展接口，停止向已登录设备拉取实时预览码流。
前置条件	已调用 CLIENT_RealPlayEx 等拉取实时预览码流接口
函数	BOOL CLIENT_StopRealPlayEx (LLONG lRealHandle);
参数	lRealHandle *[in]* 实时预览句柄，CLIENT_RealPlayEx 等拉取*实时*预览码流接口的返回值。
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	if (!CLIENT_StopRealPlayEx(g_lRealHandle)) {
	printf("CLIENT_StopRealPlayEx Failed, g_lRealHandle[%x]!Last Error[%x]\n" , g_lRealHandle, CLIENT_GetLastError());
	}

注释	无

6.15 打开录像查询接口 CLIENT_FindFile

表6-15 CLIENT_FindFile

选项	说明
接口描述	打开录像查询句柄接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	LLONG CLIENT_FindFile(LLONG lLoginID, int nChannelId, int nRecordFileType, char* cardid, LPNET_TIME time_start, LPNET_TIME time_end, BOOL bTime, int waittime);

选项	说明
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● nChannelId *[in]* 通道 ID，从 0 开始 ● nRecordFileType *[in]* 录像文件类型 不同的值对应不同的录像文件类型，具体请参见 EM_QUERY_RECORD_TYPE 枚举说明 ● cardid *[in]* 扩展参数，与 nRecordFileType 值匹配使用 ◇ EM_RECORD_TYPE_CARD：卡号 ◇ EM_RECORD_TYPE_CONDITION：卡号&&交易类型&& 交易金额(如希望跳过某字段，则相应位置为空) ◇ EM_RECORD_TYPE_CARD_PICTURE：卡号 ◇ EM_RECORD_TYPE_FIELD：FELD1&&FELD2&&FELD3&&(如希望跳过某字段，则相应位置为空) ◇ 除以上情况外，cardid 值皆为 NULL。 ● tmStart *[in]* 录像查询开始时间，具体参见 NET_TIME 结构体说明 ● tmEnd *[in]* 录像查询结束时间，具体参见 NET_TIME 结构体说明 ● bTime *[in]* 是否按时间查，该参数目前无效，建议传 FALSE ● waittime *[in]* 等待时间
返回值	成功返回录像查询句柄，失败返回 0

选项	说明

使用示例	NET_TIME StartTime = {0};
	NET_TIME StopTime = {0};
	StartTime.dwYear = 2015;
	StartTime.dwMonth = 9;
	StartTime.dwDay = 20;
	StartTime.dwHour = 0;
	StartTime.dwMinute = 0;
	StopTime.dwYear = 2015;
	StopTime.dwMonth = 9;
	StopTime.dwDay = 21;
	StopTime.dwHour = 15;
	NET_RECORDFILE_INFO netFileInfo[30] = {0};
	int nFileCount = 0;
	// 获取录像查询句柄
	if(!CLIENT_FindFile (lLoginHandle, nChannelID, (int)EM_RECORD_TYPE_ALL, NULL, &StartTime, &StopTime, FALSE, 5000))
	{
	printf("CLIENT_FindFile: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

注释	可以在回放之前先调用本接口查询录像记录，再调用 CLIENT_FindNextFile 函数逐条返回录象记录用于播放，查询完毕可以调用 CLIENT_FindClose 关闭查询句柄。

6.16 查找录像文件接口 CLIENT_FindNextFile

表6-16 CLIENT_FindNextFile

选项	说明
接口描述	查找录像文件接口
前置条件	已调用 CLIENT_FindFile 接口获取录像查询句柄
函数	int CLIENT_FindNextFile(LLONG lFindHandle, LPNET_RECORDFILE_INFO lpFindData);
参数	● lFindHandle *[in]* 录像查询句柄，对应 CLIENT_FindFile 接口的返回值 ● lpFindData *[out]* 录像文件记录缓冲，用于输出查询到的录像文件记录，具体请参见“4.11 NET_RECORDFILE_INFO 结构体说明”。
返回值	1：成功取回一条录像记录，0：录像记录已取完，-1：参数出错。

选项	说明

使用示例	NET_RECORDFILE_INFO struFileData = {0};
	int result = CLIENT_FindNextFile(lFindHandle, & struFileData);
	if(result == 1)//取回一条录像文件信息
	{
	//存储录像文件
	} elseif (result == 0) { //录像文件信息数据取完
	;
	} else { //参数出错
	printf("CLIENT_FindNextFile: failed! Error code:0x%x.\n",
	CLIENT_GetLastError());
	}

注释	调用本接口之前应先调用 CLIENT_FindFile 以打开查询句柄 每次调用取回一条录像记录信息

6.17 关闭录像查询接口 CLIENT_FindClose

表6-17 CLIENT_FindClose

选项	说明
接口描述	关闭录像查询句柄接口
前置条件	已调用 CLIENT_FindFile 接口获取录像查询句柄
函数	BOOL CLIENT_FindClose(LLONG lFindHandle);
参数	lFindHandle *[in]* 录像查询句柄 对应 CLIENT_FindFile 接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	if(!CLIENT_FindClose (lFindHandle))
	{
	printf("CLIENT_FindNextFile: failed! Error code:0x%x.\n",
	CLIENT_GetLastError());
	}

注释	调用 CLIENT_FindFile 打开查询句柄，查询完毕后必须调用本函数以关闭查询句柄，并释放占用的资源

6.18 按时间回放接口 CLIENT_PlayBackByTimeEx

表6-18 CLIENT_PlayBackByTimeEx

选项	说明
接口描述	按时间方式回放--扩展接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口

选项	说明
函数	LLONG CLIENT_PlayBackByTimeEx(LLONG lLoginID, int nChannelID, LPNET_TIME lpStartTime, LPNET_TIME lpStopTime, HWND hWnd, fDownLoadPosCallBack cbDownLoadPos, LDWORD dwPosUser, fDataCallBack fDownLoadDataCallBack, LDWORD dwDataUser);
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● nChannelID *[in]* 通道 ID，从 0 开始 ● lpStartTime *[in]* 回放开始时间，具体请参见 NET_TIME 结构体说明 ● lpStopTime *[in]*回放结束时间，具体请参见 NET_TIME 结构体说明 ● hWnd *[in]* 回放窗口 ● cbDownLoadPos *[in]* 进度回调用户参数 当 cbDownLoadPos 为 0 时，表示不回调回放数据进度；当 cbDownLoadPos 不为 0 时，回放数据进度通过 cbDownLoadPos 回调函数回调给用户，具体请参见章节 3.4 实时预览数据回调函数 fRealDataCallBackEx2 回放进度回调函数 fDownLoadPosCallBack 说明 ● dwPosUser *[in]* 用户数据 SDK 通过回放数据进度回调函数 fDownLoadPosCallBack 将该数据返回给用户，以便用户后续操作 ● fDownLoadDataCallBack *[in]* 录像数据回调函数 当 fDownLoadDataCallBack 为 0 时，表示不回调录像回放数据；当 fDownLoadDataCallBack 不为 0 时，录像回放数据通过 fDownLoadDataCallBack 回调函数回调给用户 ● dwDataUser *[in]* 用户数据 SDK 通过回放数据进度回调函数 fDownLoadDataCallBack 将该数据返回给用户，以便用户后续操作。
返回值	成功返回录像回放句柄 ID，失败返回 0

选项	说明

使用示例	// 此处示例代码基于按时间回放 playsdk 库解码
	typedef HWND (WINAPI *PROCGETCONSOLEWINDOW)();
	PROCGETCONSOLEWINDOW GetConsoleWindow;
	// 获取控制台窗口句柄
	HMODULE hKernel32 = GetModuleHandle("kernel32");
	GetConsoleWindow =
	(PROCGETCONSOLEWINDOW)GetProcAddress(hKernel32,"GetConsoleW
	indow");
	HWND hWnd = GetConsoleWindow();
	int nChannelID = 0; // 通道号
	NET_TIME stuStartTime = {0};
	stuStartTime.dwYear = 2015;
	stuStartTime.dwMonth = 9;
	stuStartTime.dwDay = 3;
	
	NET_TIME stuStopTime = {0};
	stuStopTime.dwYear = 2015;
	stuStopTime.dwMonth = 9;
	stuStopTime.dwDay = 12;
	
	g_lPlayHandle = CLIENT_PlayBackByTimeEx(g_lLoginHandle, nChannelID,
	&stuStartTime, &stuStopTime, hWnd, NULL, NULL, NULL, NULL);
	if (g_lPlayHandle == 0)
	{
	printf("CLIENT_PlayBackByTimeEx: failed! Error code: 0x%x.\n",
	CLIENT_GetLastError());
	}

注释	参数 hWnd 和 fDownLoadDataCallBack 不能同时为 NULL，否则接口调用会返回失败

6.19 开启录像回放转码接口 CLIENT_PlayBackByDataType

表6-19 CLIENT_PlayBackByDataType

选项	说明
接口描述	开启实时预览转码流接口。
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	LLON G CLIENT_PlayBackByDataType(LLONG lLoginID, const NET_IN_PLAYBACK_BY_DATA_TYPE* pstInParam, NET_OUT_PLAYBACK_BY_DATA_TYPE* pstOutParam, DWORD dwWaitTime);

选项	说明
参数	lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值。 pstInParam *[in]* 入参，具体请参见 NET_IN_PLAYBACK_BY_DATA_TYPE 结构体定义 pstOutParam *[out]* 出参，具体请参见 NET_OUT_PLAYBACK_BY_DATA_TYPE 结构体定义。 dwWaitTime *[in]* 用户期望搜索时间 用户根据自己需求合理设置该参数，由于该接口为同步接口，只有等待搜索时间到达才会从接口返回。
返回值	成功返回录像回放句柄 ID，失败返回 0

使用示例	NET_IN_PLAYBACK_BY_DATA_TYPE stuIn = {sizeof(stuIn)};
	stuIn.nChannelID = 0;
	stuIn.emDataType = EM_REAL_DATA_TYPE_MP4;
	stuIn.emAudioType = EM_AUDIO_DATA_TYPE_G711A;
	stuIn.hWnd = 0;
	stuIn.cbDownLoadPos = DownLoadPosCallBack;
	stuIn.fDownLoadDataCallBack = PlayBackDataCallBack;
	stuIn.fDownLoadDataCallBackEx = DataCallBackEx;
	stuIn.dwDataUser = NULL;
	stuIn.stStartTime.dwYear = 2023;
	stuIn.stStartTime.dwMonth = 2;
	stuIn.stStartTime.dwDay = 1;
	stuIn.stStartTime.dwHour = 11;
	stuIn.stStartTime.dwMinute = 10;
	stuIn.stStartTime.dwSecond = 12;
	
	stuIn.stStopTime.dwYear = 2023;
	stuIn.stStopTime.dwMonth = 2;
	stuIn.stStopTime.dwDay = 1;
	stuIn.stStopTime.dwHour = 11;
	stuIn.stStopTime.dwMinute = 11;
	stuIn.stStopTime.dwSecond = 12;
	
	NET_OUT_PLAYBACK_BY_DATA_TYPE stuOut = {sizeof(stuOut)};
	
	g_lPlayHandle = CLIENT_PlayBackByDataType(g_lLoginHandle, &stuIn, &stuOut, 5000);
	if (0 == g_lPlayHandle)
	{
	printf("CLIENT_PlayBackByDataType fail,error:%X\n", CLIENT_GetLastError());
	}

选项	说明
注释	无

6.20 停止录像回放接口 CLIENT_StopPlayBack

表6-20 CLIENT_StopPlayBack

选项	说明
接口描述	停止录像回放接口
前置条件	已调用 CLIENT_PlayBackByTimeEx 等接口获取录像回放句柄
函数	BOOL CLIENT_StopPlayBack(LLONG lPlayHandle);
参数	lPlayHandle *[in]* 录像回放句柄 对应 CLIENT_PlayBackByTimeEx 等接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	if (!CLIENT_StopPlayBack(g_lPlayHandle))
	{
	printf("CLIENT_StopPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

注释	用户通过 CLIENT_PlayBackByTimeEx 等接口获取录像回放句柄，通过 CLIENT_StopPlayBack 关闭录像回放句柄

6.21 获取回放 OSD 时间接口 CLIENT_GetPlayBackOsdTime

表6-21 CLIENT_GetPlayBackOsdTime

选项	说明
接口描述	获取回放 OSD 时间接口。 只有打开录像回放时的接口的参数 hWnd 有效时，本接口获取的参数才有效，否则无意义
前置条件	已调用 CLIENT_PlayBackByTimeEx 等接口获取录像回放句柄
函数	BOOL CLIENT_GetPlayBackOsdTime(LLONG lPlayHandle, LPNET_TIME lpOsdTime, LPNET_TIME lpStartTime, LPNET_TIME lpEndTime);

选项	说明
参数	• lPlayHandle *[in]* 录像回放句柄，对应 CLIENT_PlayBackByTimeEx 等接口的返回值 • lpOsdTime *[out]* OSD 的时间，具体请参见 NET_TIME 结构体说明 • lpStartTime *[out]* 回放的开始时间，具体请参见 NET_TIME 结构体说明 • lpEndTime *[out]* 回放的结束时间 具体请参见 NET_TIME 结构体说明
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	NET_TIME stuOsdTime = {0};
	NET_TIME stuStartTime = {0};
	NET_TIME stuEndTime = {0};
	
	if (!CLIENT_GetPlayBackOsdTime (g_lPlayHandle, &stuOsdTime, &stuStartTime, &stuEndTime))
	{
	printf("CLIENT_GetPlayBackOsdTime Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

注释	只有打开录像回放时的接口的参数 hWnd 有效时，本接口获取的参数才有效，否则无意义

6.22 查询时间段内录像文件接口 CLIENT_QueryRecordFile

表6-22 CLIENT_QueryRecordFile

选项	说明
接口描述	查询时间段内的所有录像文件的接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_QueryRecordFile(LLONG lLoginID, int nChannelId, int nRecordFileType, LPNET_TIME tmStart, LPNET_TIME tmEnd, char* pchCardid, LPNET_RECORDFILE_INFO nriFileinfo, int maxlen, int *filecount, int waittime=1000, BOOL bTime = FALSE);

选项	说明
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● nChannelId *[in]* 通道 ID，从 0 开始 ● nRecordFileType *[in]* 录像文件类型，不同的值对应不同的录像文件类型，具体请参见 EM_QUERY_RECORD_TYPE 枚举说明 ● tmStart *[in]* 录像查询开始时间 具体请参见 NET_TIME 结构体说明 ● tmEnd *[in]* 录像查询结束时间 具体请参见 NET_TIME 结构体说明 ● pchCardid *[in]* 扩展参数，与 nRecordFileType 值匹配使用 ◇ EM_RECORD_TYPE_CARD：卡号 ◇ EM_RECORD_TYPE_CONDITION：卡号&&交易类型&& 交易金额(如希望跳过某字段，则相应位置为空) ◇ EM_RECORD_TYPE_CARD_PICTURE：卡号 ◇ EM_RECORD_TYPE_FIELD：FELD1&&FELD2&&FELD3&&(如希望跳过某字段，则相应位置为空) ◇ 除以上情况外，cardid 值皆为 NULL。 ● nriFileinfo *[out]* 返回的录像文件信息 是一个 NET_RECORDFILE_INFO 结构数组的指针，具体请参见 NET_RECORDFILE_INFO 结构体说明 ● maxlen *[in]* nriFileinfo 缓冲的最大长度 (单位字节，建议在(100~200)*sizeof(NET_RECORDFILE_INFO)之间) ● filecount *[out]* 返回的文件个数 输出参数最大只能查到缓冲满为止的录像记录 ● waittime *[in]* 等待时间 ● bTime *[in]* 是否按时间查 该参数目前无效，建议传 FALSE
返回值	成功返回 TRUE，失败返回 FALSE

选项	说明

使用示例	NET_TIME StartTime = {0};
	NET_TIME StopTime = {0};
	StartTime.dwYear = 2015;
	StartTime.dwMonth = 9;
	StartTime.dwDay = 20;
	StartTime.dwHour = 0;
	StartTime.dwMinute = 0;
	StopTime.dwYear = 2015;
	StopTime.dwMonth = 9;
	StopTime.dwDay = 21;
	StopTime.dwHour = 15;
	NET_RECORDFILE_INFO netFileInfo[30] = {0};
	int nFileCount = 0;
	//录像文件查询
	if(!CLIENT_QueryRecordFile(lLoginHandle, nChannelID,
	(int)EM_RECORD_TYPE_ALL, &StartTime, &StopTime, NULL, &netFileInfo[0],
	sizeof(netFileInfo), &nFileCount,5000, FALSE))
	{
	printf("CLIENT_QueryRecordFile: failed! Error code: %x.\n",
	CLIENT_GetLastError());
	}

注释	在按文件回放之前需要先调用本接口查询录像记录，当根据输入的时间段查询到的录像记录信息大于定义的缓冲区大小，则只返回缓冲所能存放的录像记录，可以根据需要继续查询。

6.23 按时间下载录像接口 CLIENT_DownloadByTimeEx

表6-23 CLIENT_DownloadByTimeEx

选项	说明
接口描述	按时间下载录像扩展接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	LLONG CLIENT_DownloadByTimeEx(LLONG lLoginID, int nChannelId, int nRecordFileType, LPNET_TIME tmStart, LPNET_TIME tmEnd, char *sSavedFileName, fTimeDownLoadPosCallBack cbTimeDownLoadPos, LDWORD dwUserData, fDataCallBack fDownLoadDataCallBack, LDWORD dwDataUser, void* pReserved = NULL);

选项	说明
参数	• lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 • nChannelId *[in]* 通道号，从 0 开始 • nRecordFileType *[in]* 录像文件类型 具体请参见 EM_QUERY_RECORD_TYPE 枚举说明 • tmStart *[in]* 录像下载开始时间 具体请参见 NET_TIME 结构体说明 • tmEnd *[in]* 录像下载结束时间 具体请参见 NET_TIME 结构体说明 • sSavedFileName *[in]* 期望保存的录像文件名 建议使用全路径 • cbTimeDownLoadPos *[in]* 下载进度回调函数 具体请参见 fTimeDownLoadPosCallBack 回调函数说明 • dwUserData *[in]* 下载进度回调函数的用户数据 SDK 通过下载进度回调函数 fTimeDownLoadPosCallBack 将该数据返回给用户，以便用户后续操作 • fDownLoadDataCallBack *[in]* 下载数据回调函数 具体请参见 fDataCallBack 回调函数说明 • dwDataUser *[in]* 下载回调函数的用户数据 SDK 通过回放数据进度回调函数 fDataCallBack 将该数据返回给用户，以便用户后续操作 • pReserved *[in]* 保留参数 用于后期扩展，目前无意义，默认为 NULL
返回值	成功返回下载 ID，失败返回 0

选项	说明

使用示例	int nChannelID = 0; // 通道号
	NET_TIME stuStartTime = {0};
	stuStartTime.dwYear = 2015;
	stuStartTime.dwMonth = 9;
	stuStartTime.dwDay = 17;
	
	NET_TIME stuStopTime = {0};
	stuStopTime.dwYear = 2015;
	stuStopTime.dwMonth = 9;
	stuStopTime.dwDay = 18;
	// 开启录像下载
	// 函数形参 sSavedFileName 和 fDownLoadDataCallBack 至少有一个为有效值
	g_lDownloadHandle = CLIENT_DownloadByTimeEx(g_lLoginHandle, nChannelID, EM_RECORD_TYPE_ALL, &stuStartTime, &stuStopTime, "test.dav", TimeDownLoadPosCallBack, NULL, DataCallBack, NULL);
	if (g_lDownloadHandle == 0)
	{
	printf("CLIENT_DownloadByTimeEx: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

注释	sSavedFileName 不为空，录像数据写入到该路径对应的文件; fDownLoadDataCallBack 不为空，录像数据通过回调函数返回 下载完毕后，需要调用 CLIENT_StopDownload 接口关闭下载句柄

6.24 开启录像回放转码接口 CLIENT_DownloadByDataType

表6-24 CLIENT_DownloadByDataType

选项	说明
接口描述	开启实时预览转码流接口。
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	LLON G CLIENT_DownloadByDataType(LLONG lLoginID, const NET_IN_DOWNLOAD_BY_DATA_TYPE* pstInParam, NET_OUT_DOWNLOAD_BY_DATA_TYPE* pstOutParam, DWORD dwWaitTime);

选项	说明
参数	lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值。 pstInParam *[in]* 入参，具体请参见 NET_IN_DOWNLOAD_BY_DATA_TYPE 结构体定义 pstOutParam *[out]* 出参，具体请参见 NET_OUT_DOWNLOAD_BY_DATA_TYPE 结构体定义。 dwWaitTime *[in]* 用户期望搜索时间 用户根据自己需求合理设置该参数，由于该接口为同步接口，只有等待搜索时间到达才会从接口返回。
返回值	成功返回录像回放句柄 ID，失败返回 0

选项	说明

使用示例	NET_IN_DOWNLOAD_BY_DATA_TYPE stuIn = {sizeof(stuIn)};
	stuIn.nChannelID = 0;
	stuIn.emDataType = EM_REAL_DATA_TYPE_MP4;
	stuIn.emAudioType = EM_AUDIO_DATA_TYPE_G711A;
	stuIn.cbDownLoadPos = TimeDownLoadPosCallBack;
	stuIn.fDownLoadDataCallBack = DataCallBack;
	stuIn.dwDataUser = NULL;
	stuIn.stStartTime.dwYear = 2023;
	stuIn.stStartTime.dwMonth = 2;
	stuIn.stStartTime.dwDay = 1;
	stuIn.stStartTime.dwHour = 11;
	stuIn.stStartTime.dwMinute = 10;
	stuIn.stStartTime.dwSecond = 12;
	
	stuIn.stStopTime.dwYear = 2023;
	stuIn.stStopTime.dwMonth = 2;
	stuIn.stStopTime.dwDay = 1;
	stuIn.stStopTime.dwHour = 11;
	stuIn.stStopTime.dwMinute = 11;
	stuIn.stStopTime.dwSecond = 12;
	
	strncpy(g_szFileName, "D:\\file.mp4", sizeof(g_szFileName) - 1);
	stuIn.szSavedFileName = g_szFileName;
	
	NET_OUT_DOWNLOAD_BY_DATA_TYPE stuOut = {sizeof(stuOut)};
	g_lDownloadHandle = CLIENT_DownloadByDataType(g_lLoginHandle,
	&stuIn, &stuOut, 5000);
	if (0 == g_lDownloadHandle)
	{
	printf("CLIENT_DownloadByDataType fail,error:%X\n",
	CLIENT_GetLastError());
	}

注释	无

6.25 停止录像下载接口 CLIENT_StopDownload

表6-25 CLIENT_StopDownload

选项	说明
接口描述	停止录像下载接口
前置条件	已调用 CLIENT_DownloadByTimeEx 等录像下载接口
函数	BOOL CLIENT_StopDownload(LLONG lFileHandle);

选项	说明
参数	lFileHandle *[in]* 下载句柄 对应 CLIENT_DownloadByTimeEx 等录像下载接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	// 关闭下载，可在下载结束后调用，也可在下载中调用。
	if (g_lDownloadHandle)
	{
	if (!CLIENT_StopDownload(g_lDownloadHandle))
	{
	printf("CLIENT_StopDownload Failed, g_lDownloadHandle[%x]!Last Error[%x]\n" , g_lDownloadHandle, CLIENT_GetLastError());
	}
	}

注释	根据需要可以等文件下载完了关闭下载，也可以下载到一部分停止下载。

6.26 按文件回放接口 CLIENT_PlayBackByRecordFileEx

表6-26 CLIENT_PlayBackByRecordFileEx

选项	说明
接口描述	按文件方式回放扩展接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	LLONG CLIENT_PlayBackByRecordFileEx(LLONG lLoginID, LPNET_RECORDFILE_INFO lpRecordFile, HWND hWnd, fDownLoadPosCallBack cbDownLoadPos, LDWORD dwPosUser, fDataCallBack fDownLoadDataCallBack, LDWORD dwDataUser);

选项	说明
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● lpRecordFile *[in]* 录像文件信息 通过录像信息查询接口获得，如 CLIENT_FindNextFile 接口 具体请参见 NET_RECORDFILE_INFO 结构体说明 ● hWnd *[in]* 回放窗口 ● cbDownLoadPos *[in]* 录像进度回调函数 当 cbDownLoadPos 为 0 时，表示不回调回放数据进度；当 cbDownLoadPos 不为 0 时，回放数据进度通过 cbDownLoadPos 回调函数回调给用户，具体请参见 fDownLoadPosCallBack 回调函数说明 ● dwPosUser *[in]* 用户数据 SDK 通过回放数据进度回调函数 fDownLoadPosCallBack 将该数据返回给用户，以便用户后续操作 ● fDownLoadDataCallBack *[in]* 录像数据回调函数 当 fDownLoadDataCallBack 为 0 时，表示不回调录像回放数据；当 fDownLoadDataCallBack 不为 0 时，录像回放数据通过 fDownLoadDataCallBack 回调函数回调给用户，具体请参见 fDataCallBack 回调函数说明 ● dwDataUser *[in]* 用户数据 SDK 通过回放数据进度回调函数 fDownLoadDataCallBack 将该数据返回给用户，以便用户后续操作
返回值	成功返回录像回放句柄，失败返回 0

使用示例	// 函数形参 hWnd 需为 有效值
	// stuNetFileInfo 为 CLIENT_FindFile，CLIENT_FindNextFile，CLIENT_FindClose 三个接口组合获取的一条录像文件信息
	g_lPlayHandle = CLIENT_PlayBackByRecordFileEx(g_lLoginHandle, &stuNetFileInfo, hWnd, NULL, NULL, NULL, NULL);
	if (g_lPlayHandle == 0)
	{
	printf("CLIENT_PlayBackByRecordFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

注释	参数 hWnd 和 fDownLoadDataCallBack 不能同时为 NULL，否则接口调用会返回失败。

6.27 暂停录像回放接口 CLIENT_PausePlayBack

表6-27 CLIENT_PausePlayBack

选项	说明
接口描述	暂停或恢复录像回放 只有打开录像回放时的接口的参数 hWnd 有效时，本接口获取的参数才有效，否则无意义
前置条件	已调用 CLIENT_PlayBackByTimeEx 等接口获取录像回放句柄
函数	BOOL CLIENT_PausePlayBack(LLONG lPlayHandle, BOOL bPause);
参数	● lPlayHandle *[in]* 录像回放句柄 对应 CLIENT_PlayBackByTimeEx 等接口的返回值 ● bPause *[in]* 回放暂停与恢复播放控制标识 TRUE :暂停，FALSE :恢复
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	if (!CLIENT_ PausePlayBack (g_lPlayHandle))
	{
	printf("CLIENT_PausePlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

注释	只有打开录像回放时的接口的参数 hWnd 有效时，本接口获取的参数才有效，否则无意义

6.28 定位录像回放起始点接口 CLIENT_SeekPlayBack

表6-28 CLIENT_SeekPlayBack

选项	说明
接口描述	定位录像回放起始点
前置条件	已调用 CLIENT_PlayBackByTimeEx 等接口获取录像回放句柄
函数	BOOL CLIENT_SeekPlayBack(LLONG lPlayHandle, unsigned int offsettime, unsigned int offsetbyte);
参数	● lPlayHandle *[in]* 录像回放句柄 对应 CLIENT_PlayBackByTimeEx 等接口的返回值 ● offsettime *[in]* 相对开始处偏移时间，单位为秒 ● offsetbyte *[in]* 该字段已废除 该值需设置为 0xffffffff

选项	说明
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	int nOffsetSeconds = 2 * 60 * 60; // 拖动至 stuStartTime 后 2*60*60 秒的位置开始回放
	if (FALSE == CLIENT_SeekPlayBack (g_lPlayHandle, nOffsetSeconds, 0xffffffff))
	{
	printf("CLIENT_SeekPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

注释	无

6.29 录像快放接口 CLIENT_FastPlayBack

表6-29 CLIENT_FastPlayBack

选项	说明
接口描述	快放接口，将当前帧率提高一倍。 只有打开录像回放时的接口的参数 hWnd 有效时，本接口获取的参数才有效，否则无意义
前置条件	已调用 CLIENT_PlayBackByTimeEx 等接口获取录像回放句柄
函数	BOOL CLIENT_FastPlayBack(LLONG lPlayHandle);
参数	lPlayHandle *[in]* 录像回放句柄 对应 CLIENT_PlayBackByTimeEx 等接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	if (!CLIENT_ FastPlayBack (g_lPlayHandle))
	{
	printf("CLIENT_FastPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

注释	● 不支持无限制快放，目前最大 200 帧，大于时返回 FALSE。 ● 仅当打开录像回放接口的参数 hWnd 有效时，本接口获取的参数才有效，否则无意义

6.30 录像慢放接口 CLIENT_SlowPlayBack

表6-30 CLIENT_SlowPlayBack

选项	说明
接口描述	慢放，将当前帧率降低一倍
前置条件	已调用 CLIENT_PlayBackByTimeEx 等接口获取录像回放句柄

选项	说明
函数	BOOL CLIENT_SlowPlayBack (LLONG lPlayHandle);
参数	lPlayHandle *[in]* 录像回放句柄 对应 CLIENT_PlayBackByTimeEx 等接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	if (!CLIENT_SlowPlayBack (g_lPlayHandle))
	{
	printf("CLIENT_SlowPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

注释	最慢为每秒一帧，小于 1 则返回 FALSE。 打开录像回放时的接口的参数 hWnd 为 0，且设备支持回放速度控制情况下，SDK 向设备发送回放速度控制命令。 打开录像回放时的接口的参数 hWnd 为有效值，且设备支持回放速度控制情况下，SDK 向设备发送回放速度控制命令并对界面显示的 playsdk 库调用相应的速度控制命令。

6.31 恢复正常播放接口 CLIENT_NormalPlayBack

表6-31 CLIENT_NormalPlayBack

选项	说明
接口描述	恢复正常播放速度接口
前置条件	已调用 CLIENT_PlayBackByTimeEx 等接口获取录像回放句柄
函数	BOOL CLIENT_NormalPlayBack(LLONG lPlayHandle);
参数	lPlayHandle *[in]* 录像回放句柄 对应 CLIENT_PlayBackByTimeEx 等接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	if (!CLIENT_NormalPlayBack (g_lPlayHandle))
	{
	printf("CLIENT_NormalPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

注释	打开录像回放时的接口的参数 hWnd 为 0，且设备支持回放速度控制情况下，SDK 向设备发送回放速度控制命令。 打开录像回放时的接口的参数 hWnd 为有效值，且设备支持回放速度控制情况下，SDK 向设备发送回放速度控制命令并对界面显示的 playsdk 库调用相应的速度控制命令。

6.32 按文件下载录像接口 CLIENT_DownloadByRecordFileEx

表6-32 CLIENT_DownloadByRecordFileEx

选项	说明
接口描述	按文件下载录像扩展接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	LLONG CLIENT_DownloadByRecordFileEx(LLONG lLoginID, LPNET_RECORDFILE_INFO lpRecordFile, char *sSavedFileName, fDownLoadPosCallBack cbDownLoadPos, LDWORD dwUserData, fDataCallBack fDownLoadDataCallBack, LDWORD dwDataUser, void* pReserved = NULL);
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● lpRecordFile *[in]* 录像文件信息指针 通过录像查询接口获得，具体请参见 NET_RECORDFILE_INFO 结构体 ● sSavedFileName *[in]* 期望保存的录像文件名 建议使用全路径 ● cbDownLoadPos *[in]* 下载进度回调函数 具体请参见 fDownLoadPosCallBack 回调函数说明 ● dwUserData *[in]* 下载进度回调函数的用户数据 SDK 通过下载进度回调函数 fDownLoadPosCallBack 将该数据返回给用户，以便用户后续操作 ● fDownLoadDataCallBack *[in]* 下载数据回调函数 具体请参见 fDataCallBack 回调函数说明 ● dwUserData *[in]* 下载回调函数的用户数据 SDK 通过回放数据进度回调函数 fDataCallBack 将该数据返回给用户，以便用户后续操作 ● pReserved *[in]* 保留参数 用于后期扩展，目前无意义，默认为 NULL
返回值	成功返回下载 ID，失败返回 0

选项	说明

使用示例	// 函数形参 sSavedFileName 和 fDownLoadDataCallBack 至少有一个为有效值
	g_lDownloadHandle = CLIENT_DownloadByRecordFileEx(g_lLoginHandle, &stuNetFileInfo, "test.dav", DownLoadPosCallBack, NULL, DataCallBack, NULL);
	if (g_lDownloadHandle == 0)
	{
	printf("CLIENT_DownloadByRecordFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

注释	sSavedFileName 不为空，录像数据写入到该路径对应的文件； fDownLoadDataCallBack 不为空，录像数据通过回调函数返回。 下载完毕后，需要调用 CLIENT_StopDownload 接口关闭下载句柄。

6.33 解析配置信息接口 CLIENT_ParseData

表6-33 CLIENT_ParseData

选项	说明
接口描述	解析查询到的配置信息
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_ParseData(char* szCommand, char* szInBuffer, LPVOID lpOutBuffer, DWORD dwOutBufferSize, void* pReserved);
参数	● szCommand *[in]* 命令参数 具体请参见下面注释项 ● szInBuffer *[in]* 输入缓存 输入缓存内部存储需要解析的 json 串内容 ● lpOutBuffer *[out]* 输出缓存 不同的命令参数对应不同的结构体类型，具体请参见下面注释项 ● dwOutBufferSize *[in]* 输出缓存的大小 ● pReserved *[in]* 保留参数
返回值	成功返回 TRUE，失败返回 FALSE

选项	说明

使用示例	CFG_PTZ_PROTOCOL_CAPS_INFO stuPtzCapsInfo = {sizeof(stuPtzCapsInfo)};
	if (FALSE == CLIENT_ParseData(CFG_CAP_CMD_PTZ, pBuffer, &stuPtzCapsInfo, sizeof(stuPtzCapsInfo), NULL))
	{
	printf("CLIENT_ParseData Failed, cmd[CFG_CAP_CMD_PTZ], Last Error[%x]\n" , CLIENT_GetLastError());
	}

注释	命令参数： #define CFG_CAP_CMD_PTZ "ptz.getCurrentProtocolCaps" // 获取云台能力集(CFG_PTZ_PROTOCOL_CAPS_INFO) #define CFG_CMD_ENCODE "Encode" // 图像通道属性配置(对应 CFG_ENCODE_INFO) 更多命令参数请参考 dhconfigsdk.h 头文件

6.34 私有云台控制接口 CLIENT_DHPTZControlEx2

表6-34 CLIENT_DHPTZControlEx2

选项	说明
接口描述	私有云台控制扩展接口。支持三维快速定位、鱼眼。
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_DHPTZControlEx2(LLONG lLoginID, int nChannelID, DWORD dwPTZCommand, LONG lParam1, LONG lParam2, LONG lParam3, BOOL dwStop , void* param4 = NULL);

选项	说明
参数	• lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 • nChannelID *[in]* 操作的通道号 通道号从 0 开始 • dwPTZCommand *[in]* 球机控制命令 具体请参见 DH_PTZ_ControlType 枚举说明和 DH_EXTPTZ_ControlType 枚举说明 • lParam1 *[in]* 辅助参数 1 与其他参数配合使用，不同的控制命令有不同的参数使用组合 • lParam2 *[in]* 辅助参数 2 与其他参数配合使用，不同的控制命令有不同的参数使用组合 • lParam3 *[in]* 辅助参数 3 与其他参数配合使用，不同的控制命令有不同的参数使用组合 • dwStop *[in]* 是否停止 对云台八方向操作及镜头操作命令有效，进行其他操作时，本参数应填充 FALSE • lParam4 *[in]* 辅助参数 4，默认为 NULL 与其他参数配合使用，不同的控制命令有不同的参数使用组合
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	if (!CLIENT_DHPTZControlEx2(g_lLoginHandle, nChannelId, DH_PTZ_UP_CONTROL, 0, 0, 0, 0, FALSE, NULL))
	{
	printf("CLIENT_DHPTZControlEx2 Failed, nChoose[DH_PTZ_UP_CONTROL]!Last Error[%x]\n" , CLIENT_GetLastError());
	}

注释	关于 lParam1-4 参数，请参考《网络 SDK 开发手册》中 CLIENT_DHPTZControlEx2 接口。

6.35 新系统能力查询接口 CLIENT_QueryNewSystemInfo

表6-35 CLIENT_QueryNewSystemInfo

选项	说明
接口描述	新系统能力查询接口，查询系统能力信息（以 Json 格式，具体请参见配置 SDK）
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口

选项	说明
函数	BOOL CLIENT_QueryNewSystemInfo(LLONG lLoginID, char* szCommand, int nChannelID, char* szOutBuffer, DWORD dwOutBufferSize, int *error, int waittime=1000);
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● szCommand *[in]* 对应查询命令 具体请参见注释 ● nChannelID *[in]* 对应查询通道 通道号从 0 开始，当传 -1 时表示查询所有通道，部分命令不支持通道号为 -1 ● szOutBuffer *[in]* 用于存储数据的缓冲区 用于存储查询过来的 json 串数据 ● dwOutBufferSize *[in]* 缓冲区大小 ● error *[out]* 返回错误码 若获取失败，则 netsdk 会将对应的错误码填入该指针地址中 ● waittime *[in]* 超时时间 等待命令返回的超时时间，默认为 1000ms
返回值	成功返回 TRUE，失败返回 FALSE

选项	说明

使用示例	char* pBuffer = new char[2048];
	if (NULL == pBuffer) {
	return;
	}
	
	int nError = 0;
	if (FALSE == CLIENT_QueryNewSystemInfo(g_lLoginHandle,
	CFG_CAP_CMD_PTZ, 0, pBuffer, 2048, &nError)) {
	printf("CLIENT_QueryNewSystemInfo Failed, cmd[CFG_CAP_CMD_PTZ],
	Last Error[%x]\n" , CLIENT_GetLastError());
	if (pBuffer) {
	delete [] pBuffer;
	pBuffer = NULL;
	}
	return;
	}

注释	该接口获取过来的 json 传还需要通过 CLIENT_ParseData 接口分析，否则无法使用，CLIENT_QueryNewSystemInfo 对应能力集命令如下： #define CFG_CAP_CMD_PTZ "ptz.getCurrentProtocolCaps" // 获取云台能力集(CFG_PTZ_PROTOCOL_CAPS_INFO) 更多命令见 dhconfigsdk.h 头文件。

6.36 设置设备工作模式接口 CLIENT_SetDeviceMode

表6-36 CLIENT_SetDeviceMode

选项	说明
接口描述	设置设备语音对讲、回放和权限等工作模式接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_SetDeviceMode(LLONG lLoginID, EM_USEDEV_MODE emType, void* pValue);
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● emType *[in]* 工作模式类型 具体请参见 EM_USEDEV_MODE 枚举说明 ● pValue *[in]* 扩展参数 不同的 emType 值对应不同的扩展参数，具体请参见 EM_USEDEV_MODE 枚举说明
返回值	成功返回 TRUE，失败返回 FALSE

选项	说明

使用示例	// 设置回放时的码流类型
	int nStreamType = 0; // 0-主辅码流,1-主码流,2-辅码流
	if(!CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType)) {
	printf("CLIENT_SetDeviceMode: failed! Error code: 0x%x.\n", CLIENT_GetLastError());
	}

注释	无

6.37 异步搜索设备接口 CLIENT_StartSearchDevicesEx

表6-37 CLIENT_StartSearchDevicesEx

选项	说明
接口描述	异步搜索同网段内 IPC、NVS 等设备
前置条件	已调用 CLIENT_Init 初始化接口
函数	LLONG CLIENT_StartSearchDevicesEx (NET_IN_STARTSERACH_DEVICE* pInBuf, NET_OUT_STARTSERACH_DEVICE* pOutBuf);
参数	● pInBuf *[in]* 异步搜索设备入参，具体请参见 NET_IN_STARTSERACH_DEVICE 结构体定义 ● pOutBuf *[out]* 异步搜索设备出参，具体请参见 NET_OUT_STARTSERACH_DEVICE 结构体定义
返回值	成功返回句柄，失败返回 0

使用示例	// 开始异步搜索同网段内设备
	NET_IN_STARTSERACH_DEVICE stuInParam = {sizeof(stuInParam)};
	stuInParam.emSendType = EM_SEND_SEARCH_TYPE_BROADCAST;
	stuInParam.cbSearchDevices = SearchDevicesCBEx;
	NET_OUT_STARTSERACH_DEVICE stuOutParam = {sizeof(stuOutParam)};
	LLONG g_lSearchHandle =
	CLIENT_StartSearchDevicesEx(SearchDevicesCB, &g_lDeviceList);
	if (NULL == g_lSearchHandle)
	{
	printf("CLIENT_StartSearchDevicesEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
	return;
	}

注释	该接口只支持搜索同网段内的设备，跨网段需要调用 CLIENT_SearchDevicesByIPs 接口。

6.38 查询设备状态接口 CLIENT_QueryDevState

表6-38 CLIENT_QueryDevState

选项	说明
接口描述	查询设备状态
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_QueryDevState(LLONG lLoginID, int nType, char *pBuf, int nBufLen, int *pRetLen, int waittime=1000);
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值。 ● nType *[in]* 查询类型 具体请参见下面注释项 ● pBuf *[out]* 输出缓存区 用于存储查询到的结果信息，与查询类型匹配使用，具体请参见下面注释项 ● nBufLen *[in]* 缓存区大小 ● pRetLen *[out]* 实际查询到的数据长度，单位为字节 ● waittime *[in]* 查询等待时间，默认 1000ms
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	// 获取前端设备支持的语音对讲编码类型
	DHDEV_TALKFORMAT_LIST stulstTalkEncode;
	int retlen = 0;
	bSuccess = CLIENT_QueryDevState(g_lLoginHandle,
	DH_DEVSTATE_TALK_ECTYPE, (char*)&stulstTalkEncode,
	sizeof(stulstTalkEncode), &retlen, 3000);
	if (!(bSuccess && retlen == sizeof(stulstTalkEncode)))
	{
	printf("CLIENT_QueryDevState cmd[%d] Failed!Last Error[%x]\n" ,
	DH_DEVSTATE_TALK_ECTYPE, CLIENT_GetLastError());
	}

选项	说明
注释	支持查询类型如下： #define DH_DEVSTATE_TALK_ECTYPE　0x0009 // 查询设备支持的语音对讲格式列表，见结构体 DHDEV_TALKFORMAT_LIST 更多命令请参考 dhnetsdk.h 头文件

6.39 打开语音对讲接口 CLIENT_StartTalkEx

表6-39 CLIENT_StartTalkEx

选项	说明
接口描述	打开语音对讲扩展接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	LLONG CLIENT_StartTalkEx(LLONG lLoginID, pfAudioDataCallBack pfcb, LDWORD dwUser);
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● pfcb *[in]* 语言对讲音频数据回调函数 具体请参见 pfAudioDataCallBack 回调函数说明 ● dwUser *[in]* 语言对讲音频数据回调函数的用户数据 SDK 通过语音对讲的音频数据回调函数 pfAudioDataCallBack 将该数据返回给用户，以便用户后续操作
返回值	成功返回和设备对讲的句柄，失败返回 0

使用示例	g_lTalkHandle = CLIENT_StartTalkEx(g_lLoginHandle,
	AudioDataCallBack, (DWORD)NULL);
	if(0 == g_lTalkHandle)
	{
	printf("CLIENT_StartTalkEx Failed!Last Error[%x]\n",
	CLIENT_GetLastError());
	}

注释	无

6.40 停止语音对讲接口 CLIENT_StopTalkEx

表6-40 CLIENT_StopTalkEx

选项	说明
接口描述	停止语音对讲扩展接口
前置条件	已调用 CLIENT_StartTalkEx 等打开语音对讲接口

选项	说明
函数	BOOL CLIENT_StopTalkEx(LLONG lTalkHandle);
参数	lTalkHandle *[in]* 语言对讲的句柄 ID 对应 CLIENT_StartTalkEx 等打开语音对讲接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	if(!CLIENT_StopTalkEx(g_lTalkHandle)) {
	printf("CLIENT_StopTalkEx Failed!Last Error[%x]\n",
	CLIENT_GetLastError());
	} else {
	g_lTalkHandle = 0;
	}

注释	无

6.41 开始 PC 端录音接口 CLIENT_RecordStartEx

表6-41 CLIENT_RecordStartEx

选项	说明
接口描述	开始 PC 端录音扩展接口（对 CLIENT_RecordStart()扩展）
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_RecordStartEx(LLONG lLoginID);
参数	lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	BOOL bSuccess = CLIENT_RecordStartEx(g_lLoginHandle);
	if(!bSuccess)
	{
	printf("CLIENT_RecordStartEx Failed!Last Error[%x]\n",
	CLIENT_GetLastError());
	}

注释	无

6.42 结束 PC 端录音接口 CLIENT_RecordStopEx

表6-42 CLIENT_RecordStopEx

选项	说明
接口描述	结束 PC 端录音扩展接口（对 CLIENT_RecordStop()扩展）
前置条件	已调用 CLIENT_RecordStartEx 等打开本地录音采集接口

选项	说明
函数	BOOL CLIENT_RecordStopEx(LLONG lLoginID);
参数	lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	// 停止本地录音
	if (g_RecordFlag) {
	if (!CLIENT_RecordStopEx(g_lLoginHandle)) {
	printf("CLIENT_RecordStop Failed!Last Error[%x]\n",
	CLIENT_GetLastError());
	} else {
	g_RecordFlag = FALSE;
	}
	}

注释	CLIENT_RecordStopEx 接口需要与 CLIENT_RecordStartEx 接口配合使用

6.43 发送语音数据接口 CLIENT_TalkSendData

表6-43 CLIENT_TalkSendData

选项	说明
接口描述	发送语音数据到设备
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	LONG CLIENT_TalkSendData(LLONG lTalkHandle, char *pSendBuf, DWORD dwBufSize);
参数	● lTalkHandle *[in]* 语言对讲的句柄 ID 对应 CLIENT_StartTalkEx 等打开语音对讲接口的返回值 ● pSendBuf *[in]* 发送缓存区 存储要发送的音频数据 ● dwBufSize *[in]* 缓存大小， 是要发送的音频数据的长度，单位字节
返回值	成功时返回实际传输给设备的数据长度，失败返回-1

选项	说明

使用示例	LONG lSendLen = CLIENT_TalkSendData(lTalkHandle, pDataBuf, dwBufSize);
	if(lSendLen != (long)dwBufSize)
	{
	printf("CLIENT_TalkSendData Failed!Last Error[%x]\n" , CLIENT_GetLastError());
	}

注释	接收到 CLIENT_StartTalkEx 的回调出来的录音数据后,通过该接口发送给设备

6.44 解码音频数据接口 CLIENT_AudioDecEx

表6-44 CLIENT_AudioDecEx

选项	说明
接口描述	解码音频数据扩展接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_AudioDecEx(LLONG lTalkHandle, char *pAudioDataBuf, DWORD dwBufSize);
参数	● lTalkHandle *[in]* 语言对讲的句柄 ID 对应 CLIENT_StartTalkEx 等打开语音对讲接口的返回值 ● pAudioDataBuf *[in]* 音频缓存区 要求解码的音频数据内容 ● dwBufSize *[in]* 缓存大小， 是要求解码的音频数据内容的长度，单位字节
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	// 将收到的设备端发送过来的语音数据传给 SDK 解码播放
	if (!CLIENT_AudioDecEx(lTalkHandle, pDataBuf, dwBufSize))
	{
	printf("CLIENT_AudioDecEx Failed!Last Error[%x]\n" , CLIENT_GetLastError());
	}

注释	语音对讲中对设备传过来的数据进行解码

6.45 设置报警回调函数接口 CLIENT_SetDVRMessCallBack

表6-45 CLIENT_SetDVRMessCallBack

选项	说明
接口描述	设置报警回调函数接口

选项	说明
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	void CLIENT_SetDVRMessCallBack(fMessCallBack cbMessage, LDWORD dwUser);
参数	● cbMessage *[in]* 报警回调函数 具体请参见 fMessCallBack 回调函数说明 ● dwUser *[in]* 用户数据，SDK 通过实时预览数据回调函数 fMessCallBack 将该数据返回给用户，以便用户后续操作
返回值	无

使用示例	// 设置报警事件回调函数
	CLIENT_SetDVRMessCallBack(MessCallBack , NULL);

注释	CLIENT_SetDVRMessCallBack 接口需在报警订阅之前调用，设置的回调函数里面接收到的事件不会有包含图片的事件

6.46 订阅报警扩展接口 CLIENT_StartListenEx

表6-46 CLIENT_StartListenEx

选项	说明
接口描述	订阅报警扩展接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_StartListenEx(LLONG lLoginID);
参数	lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	// 向设备订阅报警
	if(CLIENT_StartListenEx(g_lLoginHandle))
	{
	g_bStartListenFlag = TRUE;
	printf("Listen Success!\nJust Wait Event....\n");
	}
	else
	{
	printf("CLIENT_StartListenEx Failed!Last Error[%x]\n" ,
	CLIENT_GetLastError());
	}

注释	所有设备的报警事件都是通过 CLIENT_SetDVRMessCallBack 接口设置的回调函数反馈给用户的

6.47 停止订阅报警 CLIENT_StopListen

表6-47 CLIENT_StopListen

选项	说明
接口描述	停止订阅报警
前置条件	已调用 CLIENT_StartListenEx 等报警上报订阅接口
函数	BOOL CLIENT_StopListen(LLONG lLoginID);
参数	lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	// 停止向设备订阅报警
	if (g_bStartListenFlag)
	{
	if (!CLIENT_StopListen(g_lLoginHandle))
	{
	printf("CLIENT_StopListen Failed!Last Error[%x]\n",
	CLIENT_GetLastError());
	}
	else
	{
	g_bStartListenFlag = FALSE;
	}
	}

注释	无

6.48 停止异步搜索接口 CLIENT_StopSearchDevices

表6-48 CLIENT_StopSearchDevices

选项	说明
接口描述	停止异步搜索同网段内 IPC、NVS 等设备
前置条件	已调用 CLIENT_StartSearchDevicesEx 等异步搜索设备接口
函数	BOOL CLIENT_StopSearchDevices(LLONG lSearchHandle);
参数	lSearchHandle *[in]* 异步搜索设备 ID 对应 CLIENT_StartSearchDevicesEx 等异步搜索设备接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

选项	说明

使用示例	// 停止异步搜索同网段设备
	if (NULL != g_lSearchHandle)
	{
	if (FALSE == CLIENT_StopSearchDevices(g_lSearchHandle))
	{
	printf("CLIENT_StopSearchDevices Failed!Last Error[%x]\n",
	CLIENT_GetLastError());
	}
	}

注释	该接口与 CLIENT_StartSearchDevicesEx 接口配对使用

6.49 同步搜索设备接口 CLIENT_SearchDevicesByIPs

表6-49 CLIENT_SearchDevicesByIPs

选项	说明
接口描述	同步跨网段搜索设备
前置条件	已调用 CLIENT_Init 初始化接口
函数	BOOL CLIENT_SearchDevicesByIPs(DEVICE_IP_SEARCH_INFO* pIpSearchInfo, fSearchDevicesCB cbSearchDevices, LDWORD dwUserData, char* szLocalIp, DWORD dwWaitTime);

选项	说明
参数	● pIpSearchInfo *[in]* 搜索设备信息 存储需要搜索的设备 ip，DEVICE_IP_SEARCH_INFO 参考 dhnetsdk.h 头文件 ● cbSearchDevices *[in]* 搜索设备回调函数 当有设备响应包回复过来时，SDK 将响应包解析成有效的信息，通过回调函数通知用户，具体请参见 fSearchDevicesCB 回调函数说明。回调函数不可为空 ● dwUserData *[in]* 用户数据 NetSDK 通过搜索设备回调函数 fSearchDevicesCB 将该数据返回给用户，以便用户后续操作。 ● szLocalIp *[in]* 本地 IP 可不输入，默认为 NULL。 ● dwWaitTime *[in]* 用户期望搜索时间 用户根据自己需求合理设置该参数，由于该接口为同步接口，只有等待搜索时间到达才会从接口返回。
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	DWORD dwWaitTime = 5000;
	// 用户需要注意，该接口会等到超时时间到才会返回，用户需要根据自身网络情况决定超时时间
	if (FALSE == CLIENT_SearchDevicesByIPs(&stuTmp, SearchDevicesCB, (LDWORD)&g_lDeviceList, szLocalIp, dwWaitTime))
	{
	printf("CLIENT_SearchDevicesByIPs Failed!Last Error[%x]\n", CLIENT_GetLastError());
	sreturn;
	}

注释	该接口为同步接口，只有等待搜索时间到达才会从接口返回，用户需要根据自身网络情况决定搜索时间

6.50 订阅智能图片报警接口 CLIENT_RealLoadPictureEx

表6-50 CLIENT_RealLoadPictureEx

选项	说明
接口描述	订阅智能图片报警接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口

选项	说明
函数	LLONG CLIENT_RealLoadPictureEx(LLONG lLoginID, int nChannelID, DWORD dwAlarmType, BOOL bNeedPicFile, fAnalyzerDataCallBack cbAnalyzerData, LDWORD dwUser, void* Reserved);
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● nChannelID *[in]* 智能图片报警订阅通道号，通道号从 0 开始 ● dwAlarmType *[in]* 期望订阅的报警*类型* 如：EVENT_IVS_*ALL* //所有报警信息都会上传 更多具体类型请参见 *dhnetsdk*.h 头文件 ● bNeedPicFile *[in]* 是否订阅图片文件 ◇ TRUE:表示订阅*图片*文件，在回调函数中会返回智能图片信息 ◇ FALSE:表示不订阅图片文件，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) ● cbAnalyzerData *[in]* 智能图片报警回调函数 当设备端有*智能*图片报警上报时，SDK 会调用该函数回调数据给用户 ● dwUser *[in]* 用户*数据*，SDK 通过智能图片报警回调函数 fAnalyzerDataCallBack 将该数据返回给用户，以便用户后续操作 ● Reserved *[in]* 保留参数 该字段建议填 *NULL*
返回值	失败返回 0，成功返回智能图片报警订阅 ID，将作为 CLIENT_StopLoadPic 的参数

使用示例	// 订阅智能图片报警
	LDWORD dwUser = 0;
	g_lRealLoadHandle = CLIENT_RealLoadPictureEx(g_lLoginHandle, 0, EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, dwUser, NULL);
	if (0 == g_lRealLoadHandle)
	{
	printf("CLIENT_RealLoadPictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
	return;
	}

选项	说明
注释	订阅接口每次对应一个通道，并且对应一种类型的事件 如果要订阅该通道上传所有类型的事件，可以将参数 dwAlarmType 设置为 EVENT_IVS_ALL 如果需要订阅一个通道上传两种事件，那么请调用两次 CLIENT_RealLoadPictureEx，并且传入不同的事件类型 可通过 CLIENT_StopLoadPic 接口取消订阅

6.51 设备控制扩展接口 CLIENT_ControlDeviceEx

表6-51 CLIENT_ControlDeviceEx

选项	说明
接口描述	设备控制扩展接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_ControlDeviceEx(LLONG lLoginID, CtrlType emType, void* pInBuf, void* pOutBuf = NULL, int nWaitTime = 1000);
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● emType *[in]* 控制类型 与 pInBuf 和 *pOutBuf* 匹配使用，不同的 emType，pInBuf 和 pOutBuf 指向不同的结构体，具体请参见 CtrlType 枚举说明。 ● pInBuf *[in]* 设备控制入参 与 emType 匹配使用，不同的 emType，pInBuf 指向不同的结构体，具体请参见 CtrlType 枚举说明，若 emType 对应的枚举值在枚举说明中未明确指出 pInBuf 为何结构体，则填 NULL ● pOutBuf *[out]* 设备控制出参，默认为 NULL 与 emType 匹配使用，不同的 emType，pOutBuf 指向不同的结构体，具体请参见 CtrlType 枚举说明，若 emType 对应的枚举值在枚举说明中未明确指出 pOutBuf 为何结构体，则填 NULL 当 emType 小于 0x10000 时，pOutBuf 可以不填 ● nWaitTime *[in]* 等待设备*返回的*超时时间，单位为 ms。默认为 1000ms
返回值	成功返回 TRUE，失败返回 FALSE

选项	说明

使用示例	MANUAL_SNAP_PARAMETER stuSanpParam = {0};
	stuSanpParam.nChannel = 0;
	memcpy(stuSanpParam.bySequence, "just for test",
	sizeof(stuSanpParam.bySequence) - 1);
	// 手动抓图触发报警功能，该功能只对 ITC 设备有效
	if (FALSE == CLIENT_ControlDeviceEx(g_lLoginHandle,
	DH_MANUAL_SNAP, &stuSanpParam))
	{
	printf("CLIENT_ControlDeviceEx Failed!Last Error[%x]\n",
	CLIENT_GetLastError());
	break;
	}

注释	无

6.52 取消订阅智能图片报警接口 CLIENT_StopLoadPic

表6-52 CLIENT_StopLoadPic

选项	说明
接口描述	取消智能图片报警订阅接口
前置条件	已调用 CLIENT_RealLoadPictureEx 等智能图片报警订阅接口
函数	BOOL CLIENT_StopLoadPic(LLONG lAnalyzerHandle);
参数	lAnalyzerHandle *[in]* 智能图片*报警*订阅 ID 对应 CLIENT_RealLoadPictureEx 等智能图片报警订阅接口的返回值
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	// 取消智能图片报警订阅
	if (0 != g_lRealLoadHandle)
	{
	if (FALSE == CLIENT_StopLoadPic(g_lRealLoadHandle))
	{
	printf("CLIENT_StopLoadPic Failed!Last Error[%x]\n",
	CLIENT_GetLastError());
	}
	else
	{
	g_lRealLoadHandle = 0;
	}
	}

注释	

6.53 查询录像下载进度接口 CLIENT_GetDownloadPos

表6-53 CLIENT_GetDownloadPos

选项	说明
接口描述	查询录像下载进度，单位 KB
前置条件	已调用 CLIENT_DownloadByTimeEx 等录像下载接口
函数	BOOL CLIENT_GetDownloadPos(LLONG lFileHandle, int *nTotalSize, int *nDownLoadSize);
参数	● lFileHandle *[in]* 下载句柄 对应 CLIENT_*DownloadByTimeEx* 等录像下载接口的返回值 ● nTotalSize *[out]* 下载的总*大小*，单位 KB ● nDownLoadSize *[out]* 已下载的*长度*，单位 KB
返回值	成功返回 TRUE，失败返回 FALSE

使用示例	int nTotal = 0;
	int nDownLoad = 0;
	if (FALSE == CLIENT_GetDownloadPos(g_lDownloadHandle, &nTotal, &nDownLoad))
	{
	printf("CLIENT_GetDownloadPos Failed!Last Error[%x]\n", CLIENT_GetLastError());
	}

注释	无

6.54 设置抓图回调函数接口 CLIENT_SetSnapRevCallBack

表6-54 CLIENT_SetSnapRevCallBack

选项	说明
接口描述	设置前端视频抓图回调函数接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	void CLIENT_SetSnapRevCallBack(fSnapRev OnSnapRevMessage, LDWORD dwUser);

选项	说明
参数	● OnSnapRevMessage *[in]* 前端视频抓图回调函数 具体请参见 *fSnapRev* 回调函数说明 ● dwUser *[in]* 用户数据，SDK 通过前端视频抓图回调函数 fSnapRev 将该数据返回给用户，以便用户后续操作
返回值	无

使用示例	// 设置前端视频抓图回调函数
	CLIENT_SetSnapRevCallBack(SnapRev, NULL);

注释	CLIENT_SetSnapRevCallBack 接口需在前端视频抓图接口之前调用

6.55 抓图请求接口 CLIENT_SnapPictureEx

表6-55 CLIENT_SnapPictureEx

选项	说明
接口描述	抓图请求扩展接口
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_SnapPictureEx(LLONG lLoginID, SNAP_PARAMS *par, int *reserved = 0);
参数	● lLoginID *[in]* 设备登录 ID 对应 CLIENT_LoginWithHighLevelSecurity 设备登录接口的返回值 ● par *[in]* 抓图参数 具体请参见 SNAP_PARAMS 结构体说明 ● reserved *[in]* 保留字段
返回值	成功返回 TRUE，失败返回 FALSE

选项	说明

使用示例	// 发送抓图命令给前端设备
	SNAP_PARAMS stuSnapParams;
	stuSnapParams.Channel = nChannelId;
	stuSnapParams.mode = nSnapType;
	stuSnapParams.CmdSerial = ++g_nCmdSerial; // 请求序列号，有效值范围 0～65535，超过范围会被截断为 unsigned short
	if (FALSE == CLIENT_SnapPictureEx(g_lLoginHandle, &stuSnapParams))
	{
	printf("CLIENT_SnapPictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
	return;
	}
	else
	{
	printf("CLIENT_SnapPictureEx succ\n");
	}

注释	无

6.56 启动程序 CLIENT_StartApp

表6-56 CLIENT_StartApp

选项	说明
接口描述	启动 APP 程序
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_StartApp(LLONG lLoginID, const NET_IN_START_APP* pInParam, NET_OUT_START_APP* pOutParam, int nWaitTime);
参数	● [in] lLoginID 登录句柄 ● [in] pInParam 接口输入参数，内存资源由用户申请和释放 ● [out] pOutParam 接口输出参数，内存资源由用户申请和释放 ● [in] nWaitTime 接口超时时间，单位毫秒
返回值	return TRUE 表示成功 FALSE 表示失败

使用示例	//启动程序
	NET_IN_START_APP stuIn = {sizeof(NET_IN_START_APP)};
	memset(&stuIn, 0, sizeof(NET_IN_START_APP));
	stuIn.dwSize = sizeof(NET_IN_START_APP);
	NET_OUT_START_APP stuOut = {sizeof(NET_OUT_START_APP)};
	memset(&stuOut, 0, sizeof(NET_OUT_START_APP));
	stuOut.dwSize = sizeof(NET_OUT_START_APP);
	stuIn.nAppID = 3214;
	strncpy(stuIn.szAppName, "TestAppCheck", sizeof(stuIn.szAppName) - 1);
	BOOL bRet = CLIENT_StartApp(lLoginID, &stuIn,&stuOut,5000);

注释	无

6.57 查询配置信息 CLIENT_GetNewDevConfig

表6-57 CLIENT_GetNewDevConfig + CFG_CMD_DEV_GENERRAL

选项	说明
接口描述	查询配置信息
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_GetNewDevConfig(LLONG lLoginID, char* szCommand, int nChannelID, char* szOutBuffer, DWORD dwOutBufferSize, int *error, int waittime=500, void* pReserved = NULL);
参数	● [in] lLoginID 登录句柄 ● [in] szCommand 配置名 ● [in] nChannelID 通道号 ● [out] szOutBuffer 配置信息， json 格式 ● [in] dwOutBufferSize 配置信息原始有效程度 ● [out] error 错误码 ● [in] waittime 接口超时时间，单位毫秒 ● [in] pReserved 类型为 NET_CONFIG_RESERVED_PARA 指针
返回值	return TRUE 表示成功 FALSE 表示失败

使用示例	LLONG lLoginID = *(LLONG*)data;
	int nChannel = -1;
	int nBufLen = 1024*1024*20;
	char *pbufsize = new char[nBufLen];
	memset(pbufsize, 0, nBufLen);
	int nError = 0;
	int nRetlen = 0;
	CFG_DEV_DISPOSITION_INFO *pstinfo = new CFG_DEV_DISPOSITION_INFO;
	memset(pstinfo, 0, sizeof(CFG_DEV_DISPOSITION_INFO));
	BOOL bRet = CLIENT_GetNewDevConfig(lLoginID, CFG_CMD_DEV_GENERRAL, nChannel, pbufsize, nBufLen, &nError, 5000);

注释	无

6.58 获取安装的应用列表 CLIENT_GetInstalledAppInfo

表6-58 CLIENT_GetInstalledAppInfo

选项	说明
接口描述	获取安装的应用列表
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口

选项	说明
函数	BOOL CLIENT_GetInstalledAppInfo(LLONG lLoginID, const NET_IN_GET_INSTALLED_APP_INFO* pInParam, NET_OUT_GET_INSTALLED_APP_INFO* pOutParam, int nWaitTime);
参数	● [in] lLoginID 登录句柄 ● [in] pInParam 接口输入参数，内存资源由用户申请和释放 ● [out] pOutParam 接口输出参数，内存资源由用户申请和释放 ● [in] nWaitTime 接口超时时间，单位毫秒
返回值	return TRUE 表示成功 FALSE 表示失败

使用示例	LLONG lLoginID = *(LLONG*)data;
	NET_IN_GET_INSTALLED_APP_INFO stuIn = {sizeof(NET_IN_GET_INSTALLED_APP_INFO)};;
	memset(&stuIn, 0, sizeof(NET_IN_GET_INSTALLED_APP_INFO));
	stuIn.dwSize = sizeof(NET_IN_GET_INSTALLED_APP_INFO);
	NET_OUT_GET_INSTALLED_APP_INFO stuOut = {sizeof(NET_OUT_GET_INSTALLED_APP_INFO)};;
	memset(&stuOut, 0, sizeof(NET_OUT_GET_INSTALLED_APP_INFO));
	stuOut.dwSize = sizeof(NET_OUT_GET_INSTALLED_APP_INFO);
	BOOL bRet = CLIENT_GetInstalledAppInfo(lLoginID, &stuIn,&stuOut,5000);

注释	无

6.59 升级准备 CLIENT_UpgraderInstall

表6-59 CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_PREPAREEX

选项	说明
接口描述	升级准备
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_UpgraderInstall(LLONG lLoginID, EM_NET_UPGRADE_INSTALL_TYPE emType, void* pInBuf, void* pOutBuf, int nWaittime);
参数	● [in] lLoginID 登录句柄 ● [in] emType 操作类型 ● [in] pInBuf 内存由用户申请释放，emType 类型对应的结构体 ● [out] pOutBuf 内存由用户申请释放，emType 类型对应的结构体 ● [in] nWaittime 接口超时时间，单位毫秒
返回值	return TRUE 表示成功 FALSE 表示失败

选项	说明

使用示例	LLONG lLoginID = *(LLONG*)data;
	EM_NET_UPGRADE_INSTALL_TYPE emType = EM_UPGRADER_INSTALL_PREPAREEX;
	NET_IN_INSTALL_PREPAREEX stuIn = {sizeof(NET_IN_INSTALL_PREPAREEX)};
	memset(&stuIn, 0, sizeof(NET_IN_INSTALL_PREPAREEX));
	stuIn.dwSize = sizeof(NET_IN_INSTALL_PREPAREEX);
	NET_OUT_INSTALL_PREPAREEX stuOut = {sizeof(NET_OUT_INSTALL_PREPAREEX)};
	memset(&stuOut, 0, sizeof(NET_OUT_INSTALL_PREPAREEX));
	stuOut.dwSize = sizeof(NET_OUT_INSTALL_PREPAREEX);
	stuIn.bReliable = true;
	stuIn.emNextOperate = EM_NET_NEXT_OPERATE_INSTALL;
	stuIn.nTotalLength = 1024;
	strncpy(stuIn.szAppName, "TestAppCheck", sizeof(stuIn.szAppName) - 1);
	BOOL bRet = CLIENT_UpgraderInstall(lLoginID, emType,&stuIn,&stuOut,5000);

注释	无

6.60 推送升级数据 CLIENT_UpgraderInstall

表6-60 CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_APPEND_DATA

选项	说明
接口描述	推送升级数据
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_UpgraderInstall(LLONG lLoginID, EM_NET_UPGRADE_INSTALL_TYPE emType, void* pInBuf, void* pOutBuf, int nWaittime);
参数	● [in] lLoginID 登录句柄 ● [in] emType 操作类型 ● [in] pInBuf 内存由用户申请释放，emType 类型对应的结构体 ● [out] pOutBuf 内存由用户申请释放，emType 类型对应的结构体 ● [in] nWaittime 接口超时时间，单位毫秒
返回值	return TRUE 表示成功 FALSE 表示失败

选项	说明

使用示例	LLONG lLoginID = *(LLONG*)data;
	EM_NET_UPGRADE_INSTALL_TYPE emType = EM_UPGRADER_INSTALL_APPEND_DATA;
	NET_IN_INSTALL_APPEND_DATA stuIn = {sizeof(NET_IN_INSTALL_APPEND_DATA)};
	memset(&stuIn, 0, sizeof(NET_IN_INSTALL_APPEND_DATA));
	stuIn.dwSize = sizeof(NET_IN_INSTALL_APPEND_DATA);
	NET_OUT_INSTALL_APPEND_DATA stuOut = {sizeof(NET_OUT_INSTALL_APPEND_DATA)};
	memset(&stuOut, 0, sizeof(NET_OUT_INSTALL_APPEND_DATA));
	stuOut.dwSize = sizeof(NET_OUT_INSTALL_APPEND_DATA);
	stuIn.nTotalLen = 1024;
	stuIn.nAppendLen = 32;
	stuIn.pAppendData = new unsigned char[stuIn.nAppendLen];
	for(int i = 0;i<stuIn.nAppendLen;i++)
	{
	stuIn.pAppendData[i] = (unsigned char)(i+1);
	}
	BOOL bRet = CLIENT_UpgraderInstall(lLoginID, emType,&stuIn,&stuOut,5000);

注释	无

6.61 执行升级 CLIENT_UpgraderInstall

表6-61 CLIENT_UpgraderInstall + EM_UPGRADER_INSTALL_EXECUTE

选项	说明
接口描述	执行升级
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_UpgraderInstall(LLONG lLoginID, EM_NET_UPGRADE_INSTALL_TYPE emType, void* pInBuf, void* pOutBuf, int nWaittime);
参数	● [in] lLoginID 登录句柄 ● [in] emType 操作类型 ● [in] pInBuf 内存由用户申请释放，emType 类型对应的结构体 ● [out] pOutBuf 内存由用户申请释放，emType 类型对应的结构体 ● [in] nWaittime 接口超时时间，单位毫秒
返回值	return TRUE 表示成功 FALSE 表示失败

选项	说明

使用示例	LLONG lLoginID = *(LLONG*)data;
	EM_NET_UPGRADE_INSTALL_TYPE emType = EM_UPGRADER_INSTALL_EXECUTE;
	NET_IN_INSTALL_EXECUTE stuIn = {sizeof(NET_IN_INSTALL_EXECUTE)};
	memset(&stuIn, 0, sizeof(NET_IN_INSTALL_EXECUTE));
	stuIn.dwSize = sizeof(NET_IN_INSTALL_EXECUTE);
	NET_OUT_INSTALL_EXECUTE stuOut = {sizeof(NET_OUT_INSTALL_EXECUTE)};
	memset(&stuOut, 0, sizeof(NET_OUT_INSTALL_EXECUTE));
	stuOut.dwSize = sizeof(NET_OUT_INSTALL_EXECUTE);
	stuIn.bAutoReboot = false;
	BOOL bRet = CLIENT_UpgraderInstall(lLoginID, emType,&stuIn,&stuOut,5000);

注释	无

6.62 停止程序运行 CLIENT_StopApp

表6-62 CLIENT_StopApp

选项	说明
接口描述	停止程序运行
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_StopApp(LLONG lLoginID, const NET_IN_STOP_APP* pInParam, NET_OUT_STOP_APP* pOutParam, int nWaitTime);
参数	● [in] lLoginID 登录句柄 ● [in] pInParam 接口输入参数，内存资源由用户申请和释放 ● [out] pOutParam 接口输出参数，内存资源由用户申请和释放 ● [in] nWaitTime 接口超时时间，单位毫秒
返回值	return TRUE 表示成功 FALSE 表示失败

使用示例	LLONG lLoginID = *(LLONG*)data;
	NET_IN_STOP_APP stuIn = {sizeof(NET_IN_STOP_APP)};;
	memset(&stuIn, 0, sizeof(NET_IN_STOP_APP));
	stuIn.dwSize = sizeof(NET_IN_STOP_APP);
	NET_OUT_STOP_APP stuOut = {sizeof(NET_OUT_STOP_APP)};;
	memset(&stuOut, 0, sizeof(NET_OUT_STOP_APP));
	stuOut.dwSize = sizeof(NET_OUT_STOP_APP);
	stuIn.nAppID = 3214;
	strncpy(stuIn.szAppName, "TestAppCheck", sizeof(stuIn.szAppName) - 1);
	BOOL bRet = CLIENT_StopApp(lLoginID, &stuIn,&stuOut,5000);

注释	无

6.63 卸载程序 CLIENT_RemoveApp

表6-63 CLIENT_RemoveApp

选项	说明
接口描述	卸载程序
前置条件	已调用 CLIENT_LoginWithHighLevelSecurity 登录设备接口
函数	BOOL CLIENT_RemoveApp(LLONG lLoginID, const NET_IN_REMOVE_APP* pInParam, NET_OUT_REMOVE_APP* pOutParam, int nWaitTime);
参数	● [in] lLoginID 登录句柄 ● [in] pInParam 接口输入参数，内存资源由用户申请和释放 ● [out] pOutParam 接口输出参数，内存资源由用户申请和释放 ● [in] nWaitTime 接口超时时间，单位毫秒
返回值	return TRUE 表示成功 FALSE 表示失败

使用示例	LLONG lLoginID = *(LLONG*)data;
	NET_IN_REMOVE_APP stuIn = {sizeof(NET_IN_REMOVE_APP)};
	memset(&stuIn, 0, sizeof(NET_IN_REMOVE_APP));
	stuIn.dwSize = sizeof(NET_IN_REMOVE_APP);
	NET_OUT_REMOVE_APP stuOut = {sizeof(NET_OUT_REMOVE_APP)};;
	memset(&stuOut, 0, sizeof(NET_OUT_REMOVE_APP));
	stuOut.dwSize = sizeof(NET_OUT_REMOVE_APP);
	stuIn.nAppID = 3214;
	strncpy(stuIn.szAppName, "TestAppCheck", sizeof(stuIn.szAppName) - 1);
	BOOL bRet = CLIENT_RemoveApp(lLoginID, &stuIn,&stuOut,5000);

注释	无

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。