

NetSDK 编程指导手册

（智能 AI 分册）

V1.0.0

商标声明

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

责任声明

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

隐私保护提醒

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

关于本文档

- 产品请以实物为准，本文档仅供参考
- 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。
- 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。
- 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。

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。

本文档描述了 IVSS、NVR、IPC、ITC、客流量统计设备、闸机等设备的智能业务涉及的 SDK 接口以及调用流程，更多功能接口、结构体等请参见《网络 SDK 开发手册.chm》。


本文档提供的示例代码仅为演示接口调用方法，不保证能直接拷贝编译。

读者对象

使用 SDK 的软件开发工程师，产品经理，项目经理。

符号约定

在本文档中可能出现下列标志，代表的含义如下。

标识	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

编号	版本号	修订内容	发布日期
1	V1.0.0	首次发布	2018.10.26

以下对本文档中使用的专业名词分别说明，帮助您更好的理解各个业务功能，请参见表 1-1。

表1-1 名词解释

名词	解释
IVSS	Intelligent Video Surveillance System，区别传统 NVR 只进行存储的业务形态，加入智能化分析，形成一体化管理的系统。
人脸检测	通过对视频进行智能分析，检测出其中的人脸及人脸的特征信息（年龄、性别，表情等）。
人脸识别	包含人脸检测，通过对视频进行智能分析，根据布控的人脸库，检测出视频中的人脸是否在布控的人脸库中。
历史库	用于存放设备抓拍到的人脸图的库。
人脸库	通过预先导入一些人脸图片到 IVSS、NVR、前端 IPC 等设备中，实时检测人脸是否在其中。
以通道为对象进行布控	对某一通道进行一个或者多个人脸库的布控与撤控，应用场景：该通道检测到的人脸与布控库进行比对，并给出人脸识别的结果。属于人脸库布控的一种方式。
以库为对象进行布控	将某个人脸库布控到一个或者多个通道上，应用场景：该人脸库布控的通道检测到的人脸与该人脸库进行比对，对比完成后，给出人脸识别的结果。属于人脸库布控的一种方式。
以图搜图	外部导入一张图片和相似度值，IVSS、NVR 等设备通过这张图检索历史库或人脸库是否已经存在匹配的人脸，并返回在相似度之上的图片结果。
ITC	Intelligent Traffic Camera，智能交通摄像机。具有抓拍车辆图片并自动分析交通事件的功能。
绊线入侵检测	自动检测穿越警戒线的行为。
区域入侵检测	自动检测目标入侵警戒区的行为，包括“穿越区域”和“在区域内”。
客流量	相机标定区域内的人数信息。
人数统计	实时统计相机标定区域进入或者离开的人数。
区域内人数统计	实时统计相机标定区域内的人数。

法律声明	I
前言	II
名词解释	III
1 内容简介	1
1.1 概述	1
1.2 适用性	2
1.3 应用场景	2
1.3.1 人脸检测/人脸识别/人体检测	2
1.3.2 客流量统计	4
1.3.3 智能交通	4
1.3.4 通用行为	6
1.3.5 闸机	7
2 通用功能	9
2.1 SDK 初始化	9
2.1.1 简介	9
2.1.2 接口总览	9
2.1.3 流程说明	9
2.1.4 示例代码	10
2.2 设备登录	11
2.2.1 简介	11
2.2.2 接口总览	11
2.2.3 流程说明	11
2.2.4 示例代码	13
2.3 实时监控	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	17
2.4 智能事件订阅	18
2.4.1 简介	18
2.4.2 接口总览	18
2.4.3 流程说明	18
2.4.4 示例代码	20
2.5 查询/回放/下载录像和图片	20
2.5.1 简介	20
2.5.2 接口总览	21
2.5.3 流程说明	21
2.5.4 示例代码	23
3 人脸检测与人脸识别	29
3.1 人脸事件订阅	29
3.2 人脸库的增删改查	29

3.2.1 简介	29
3.2.2 接口总览	29
3.2.3 流程说明	29
3.2.4 示例代码	31
3.3 人脸的增删改查	33
3.3.1 简介	33
3.3.2 接口总览	33
3.3.3 流程说明	33
3.3.4 示例代码	35
3.4 以通道或者库为对象布控	37
3.4.1 简介	37
3.4.2 接口总览	37
3.4.3 流程说明	38
3.4.4 示例代码	39
3.5 以图搜图	41
3.5.1 简介	41
3.5.2 接口总览	41
3.5.3 流程说明	41
3.5.4 示例代码	43
3.6 人脸相关录像图片查询与下载	45
4 人体检测	46
4.1 人体事件订阅	46
4.2 人体图片搜索	46
4.2.1 简介	46
4.2.2 接口总览	46
4.2.3 流程说明	46
4.2.4 示例代码	47
5 客流量统计	49
5.1 客流量事件订阅	49
5.1.1 简介	49
5.1.2 接口总览	49
5.1.3 流程说明	49
5.1.4 示例代码	50
5.2 客流量事件报警	51
5.3 客流量历史数据查询	51
5.3.1 简介	51
5.3.2 接口总览	51
5.3.3 流程说明	51
5.3.4 示例代码	52
6 通用行为事件	54
6.1 通用行为事件的订阅	54
6.2 通用行为事件录像查询和下载	54
7 智能交通	55
7.1 智能交通事件订阅	55
7.2 车流量历史数据查询	55
7.2.1 简介	55
7.2.2 接口总览	55

7.2.3 流程说明	55
7.2.4 示例代码	57
7.3 车辆黑白名单增删改查	58
7.3.1 简介	58
7.3.2 接口总览	59
7.3.3 流程说明	59
7.3.4 示例代码	61
7.4 车辆相关图片的查询与下载	63
7.4.1 简介	63
7.4.2 接口总览	63
7.4.3 流程说明	63
7.4.4 示例代码	65
8 闸机	67
8.1 门禁事件订阅	67
8.2 门禁卡信息管理	67
8.2.1 简介	67
8.2.2 接口总览	67
8.2.3 流程说明	67
8.2.4 示例代码	69
8.3 人脸管理	73
8.3.1 简介	73
8.3.2 接口总览	73
8.3.3 流程说明	73
8.3.4 示例代码	75
8.4 门禁进出记录查询	77
8.4.1 简介	77
8.4.2 接口总览	77
8.4.3 流程说明	78
8.4.4 示例代码	80
9 接口函数	81
9.1 SDK 初始化	81
9.1.1 SDK 初始化 CLIENT_Init	81
9.1.2 SDK 清理 CLIENT_Cleanup	81
9.1.3 设置断线重连回调函数 CLIENT_SetAutoReconnect	81
9.1.4 设置网络参数 CLIENT_SetNetworkParam	82
9.2 设备登录	82
9.2.1 用户登录设备 CLIENT_LoginEx2	82
9.2.2 用户登出设备 CLIENT_Logout	83
9.3 实时监视	83
9.3.1 打开监视 CLIENT_RealPlayEx	83
9.3.2 关闭监视 CLIENT_StopRealPlayEx	84
9.3.3 保存监视数据 CLIENT_SaveRealData	84
9.3.4 停止保存监视数据 CLIENT_StopSaveRealData	85
9.3.5 设置监视数据回调 CLIENT_SetRealDataCallBackEx	85
9.4 智能事件订阅	85
9.4.1 智能事件订阅 CLIENT_RealLoadPictureEx	85
9.4.2 智能事件的取消订阅 CLIENT_StopLoadPic	86

9.5	智能事相关录像图片查询与下载	87
9.5.1	按条件查询图片或录像 CLIENT_FindFileEx	87
9.5.2	获取查询到文件个数 CLIENT_GetTotalFileCount.....	87
9.5.3	查找文件 CLIENT_FindNextFileEx.....	88
9.5.4	结束文件查找 CLIENT_FindCloseEx	88
9.5.5	开始录像回放 CLIENT_PlayBackByTimeEx2.....	88
9.5.6	结束录像回放 CLIENT_StopPlayBack.....	89
9.5.7	开始录像下载 CLIENT_DownloadByTimeEx	89
9.5.8	停止录像下载 CLIENT_StopDownload	89
9.5.9	下载图片 CLIENT_DownloadRemoteFile	90
9.6	人脸事件订阅	90
9.7	人脸库的增删改查	90
9.7.1	人脸库的增删改操作 CLIENT_OperateFaceRecognitionGroup.....	90
9.7.2	人脸库信息查询 CLIENT_FindGroupInfo	91
9.8	人脸的增删改查	91
9.8.1	人脸的增删改操作 CLIENT_OperateFaceRecognitionDB	91
9.8.2	设置人脸的查询条件 CLIENT_OperateFaceRecognitionDB	92
9.8.3	查询人脸 CLIENT_DoFindFaceRecognition	92
9.8.4	结束人脸查找 CLIENT_StopFindFaceRecognition.....	92
9.8.5	以库为对象进行布控 CLIENT_FaceRecognitionPutDisposition.....	93
9.8.6	以库为对象进行撤控 CLIENT_FaceRecognitionDelDisposition	93
9.8.7	以通道为对象进行布控 CLIENT_SetGroupInfoForChannel.....	94
9.8.8	订阅人脸查询进度 CLIENT_AttachFaceFindState.....	94
9.8.9	取消订阅人脸查询进度 CLIENT_DetachFaceFindState	94
9.9	人体检测	95
9.9.1	人体图片下载 CLIENT_DownloadRemoteFile	95
9.10	客流量统计	95
9.10.1	客流量事件订阅 CLIENT_AttachVideoStatSummary.....	95
9.10.2	取消订阅客流量事件 CLIENT_DetachVideoStatSummary	95
9.10.3	开始查询客流历史数据（设置查询条件）CLIENT_StartFindNumberStat	96
9.10.4	查询客流历史数据 CLIENT_DoFindNumberStat.....	96
9.10.5	结束查询历史数据 CLIENT_StopFindNumberStat.....	96
9.11	智能交通	97
9.11.1	开始查找数据（设置查询条件）CLIENT_FindRecord.....	97
9.11.2	查询数据总数 CLIENT_QueryRecordCount	97
9.11.3	查询指定条数数据 CLIENT_FindNextRecord.....	98
9.11.4	结束车流量查询 CLIENT_FindRecordClose.....	98
9.11.5	黑白名单的增删改 CLIENT_OperateTrafficList.....	98
9.11.6	批量下载文件 CLIENT_DownloadMultiFile	99
9.11.7	停止批量下载文件 CLIENT_StopLoadMultiFile	99
9.12	门禁	99
9.12.1	开始查找数据（设置查询条件）CLIENT_FindRecord.....	99
9.12.2	查询指定条数数据 CLIENT_FindNextRecord.....	100
9.12.3	结束查询 CLIENT_FindRecordClose	100
9.12.4	人员/门禁记录信息操作 CLIENT_FindRecordClose	100
9.12.5	人脸图片的操作 CLIENT_FindRecordClose	101
10	回调函数定义	102

10.1	断线回调函数 fDisConnect	102
10.2	断线重连回调函数 fHaveReConnect.....	102
10.3	实时监视数据回调函数 fRealDataCallBackEx	103
10.4	智能事件回调函数 fAnalyzerDataCallBack	103
10.5	回放及按文件下载进度回调函数 fDownloadPosCallBack.....	104
10.6	回放及下载数据回调函数 fDataCallBack.....	104
10.7	人脸查询进度回调函数 fFaceFindState	105
10.8	客流量事件订阅回调 fVideoStatSumCallBack	105
10.9	批量下载文件进度回调函数 fMultiFileDownLoadPosCB	106

1.1 概述

本文档主要介绍 SDK 接口参考信息，包括主要功能、接口函数和回调函数。

主要功能包括通用功能、人脸检测和人脸识别、人体检测、客流量统计、通用行为事件、智能交通和闸机。

根据环境不同，开发包包含的文件会不同，具体如下所示。

- Windows 开发包所包含的文件，请参见表 1-1。

表1-1 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	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
	fisheye.dll	鱼眼矫正库
avnetsdk.dll 的辅助库	Infra.dll	功能辅助库
	json.dll	功能辅助库
	NetFramework.dll	功能辅助库
	Stream.dll	功能辅助库
	StreamSvr.dll	功能辅助库
dhnetsdk 辅助库	IvsDrawer.dll	图像显示库

- Linux 开发包所包含的文件，请参见表 1-2。

表1-2 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	头文件
	libdhconfigsdk.so	库文件
libavnetsdk.so 的辅助库	libInfra.so	功能辅助库
	libJson.so	功能辅助库
	libNetFramework.so	功能辅助库

库类型	库文件名称	库文件说明
	libStream.so	功能辅助库



- 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、配置以及获取和处理码流数据等。
- 配置库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。
- 辅助库用于监视、回放、对讲等功能的音视频码流解码以及本地音频采集。
- 如果功能库包含 avnetsdk，对应辅助库则是必备的。

1.2 适用性

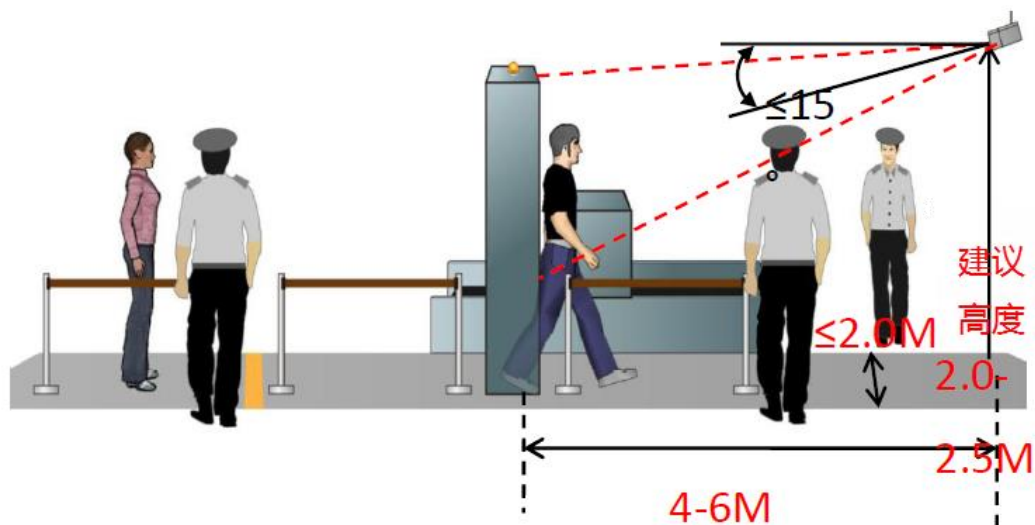
- 推荐内存：不低于 512M。
- SDK 支持的系统如下：
 - ◇ Windows
Windows 10/Windows 8.1/Windows 7/vista/2000 以及 Windows Server 2008/2003。
 - ◇ Linux
Red Hat/SUSE 等通用 Linux 系统。

1.3 应用场景

1.3.1 人脸检测/人脸识别/人体检测

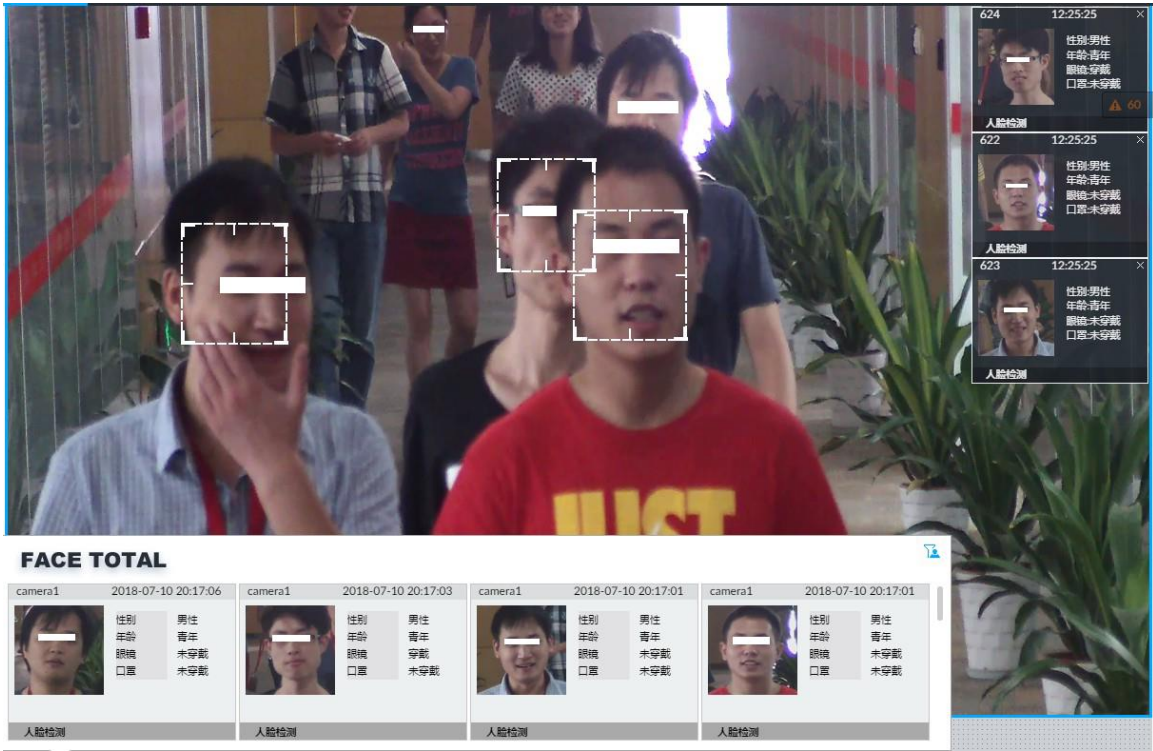
人脸检测、人脸识别和人体识别设备的使用场景如图 1-1 所示。

图1-1 人脸识别



人脸检测场景，如图 1-2 所示。

图1-2 人脸检测场景



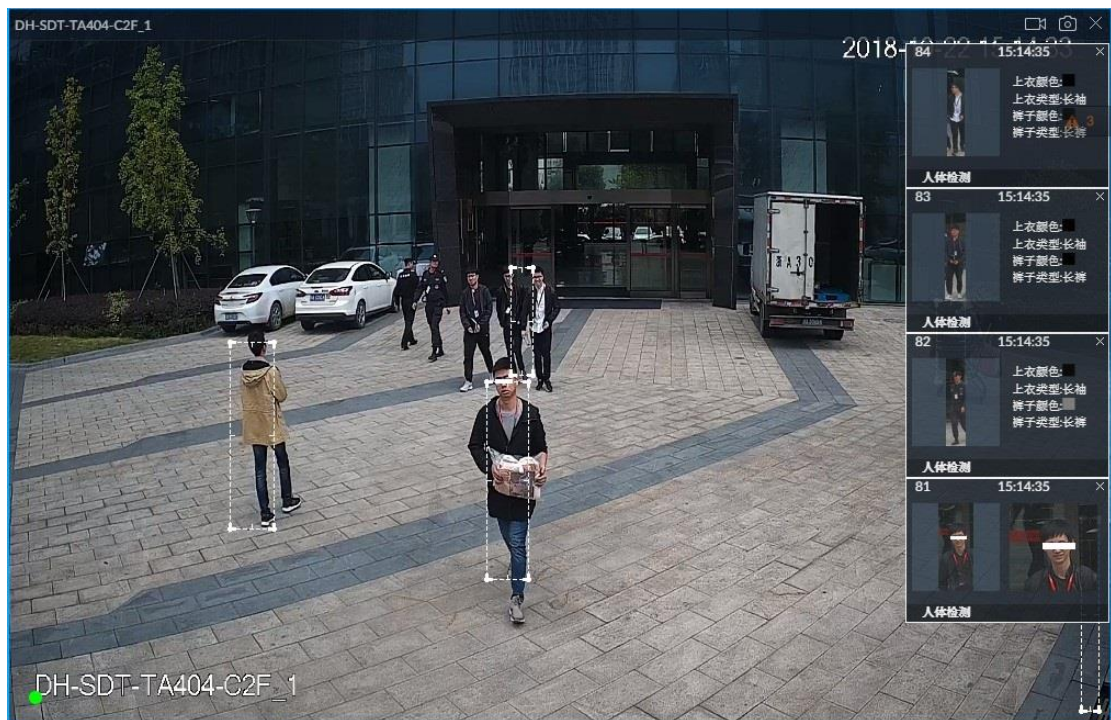
人脸识别场景，如图 1-3 所示。

图1-3 人脸识别场景



人体检测场景，如图 1-4 所示。

图1-4 人体检测场景



1.3.2 客流量统计

客流产品在实际场景的应用，如图 1-5 所示。

图1-5 客流场景



1.3.3 智能交通

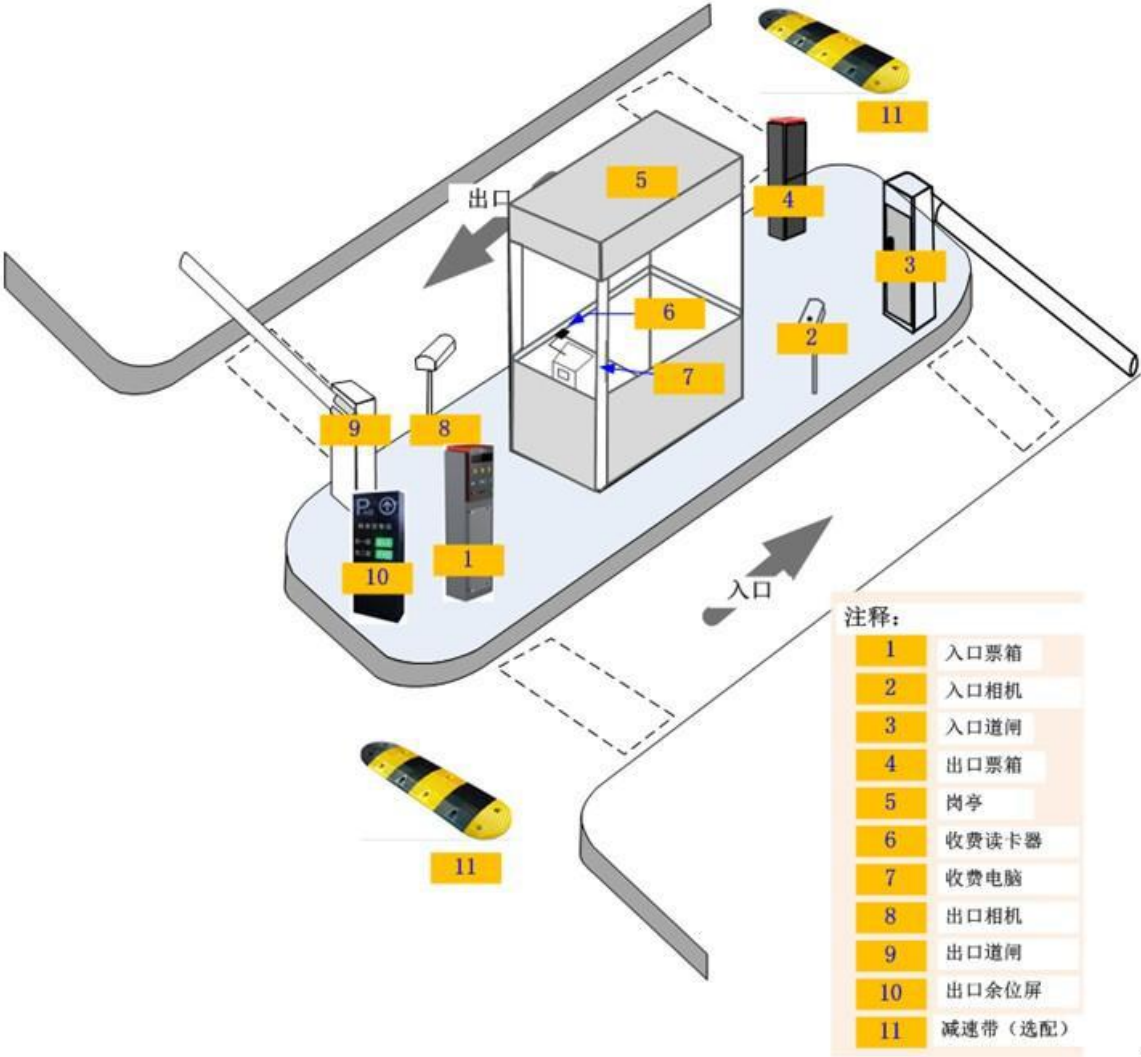
ITC 在交通路口中的应用，用于抓拍交通违法行为及车辆流量统计，如图 1-6 所示。

图1-6 ITC 在交通路口的应用



ITC 在停车场出入口的应用，用于控制车辆进出停车场及监控车位是否有空余，如图 1-7 所示。

图1-7 ITC 在停车场出入口的应用



1.3.4 通用行为

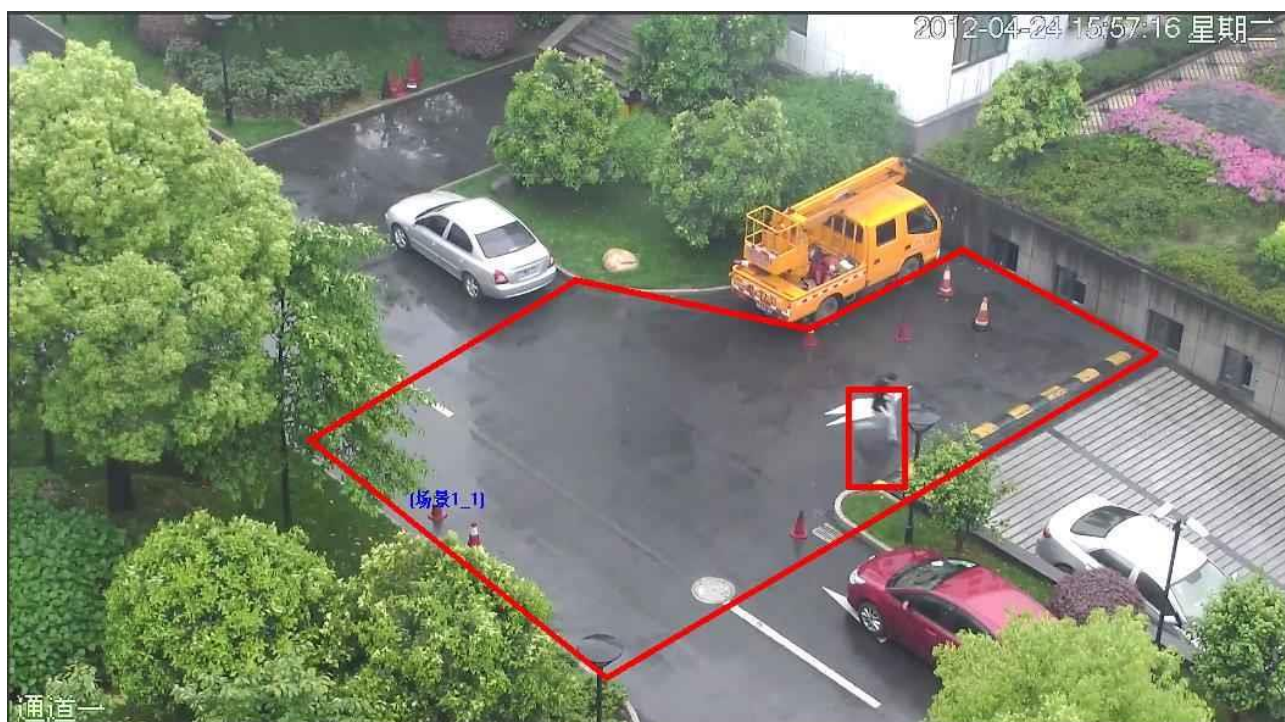
指人或者车辆穿越规则线（绊线入侵）或者入侵警戒区（区域入侵）产生相应的报警事件，同时可以区分出目标物体（人或者车辆）。

通用行为应用场景如图 1-8、1-9 所示。

图1-8 通用行为场景-绊线入侵



图1-9 通用行为场景-区域入侵



1.3.5 闸机

门禁闸机主要应用园区、景区、学校、小区、办公楼等。将采集的人脸照片和人员信息上传平台后，再由平台将数据下发到闸机系统。

门禁闸机外观如图 1-10 所示。

图1-10 摆闸外观



闸机可以通过人脸或者刷卡的方式进行开门，人脸开门如图 1-11 所示。

图1-11 人脸开门



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，但会设置一些影响全局业务的参数。

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

2.1.2 接口总览

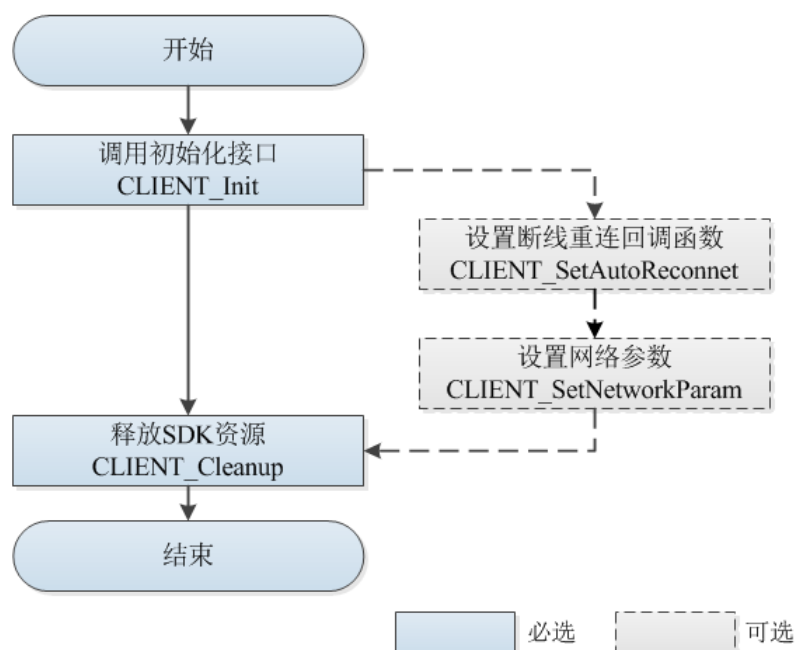
表2-1 SDK 初始化接口信息

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

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 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 （可选）调用 CLIENT_SetAutoReconnect 设置断线重连回调函数，设置后 SDK 内部断线自动重连。
- 步骤3 （可选）调用 CLIENT_SetNetworkParam 设置网络登录参数，参数中包含登录设备超时时间和尝试次数。
- 步骤4 SDK 所有功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

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

2.1.4 示例代码

```

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

```
if (FALSE == bNetSDKInitFlag)
{
    printf("Initialize client SDK fail; \n");
    return -1;
}
// 清理初始化资源
if (TRUE == bNetSDKInitFlag)
{
    CLIENT_Cleanup();
}
```

2.2 设备登录

2.2.1 简介

设备登录，即用户鉴权，是进行其他业务的前提。
用户登录设备产生唯一的登录 ID，其他功能的 SDK 接口需要传入登录 ID 才可执行。登出设备后，登录 ID 失效。

2.2.2 接口总览

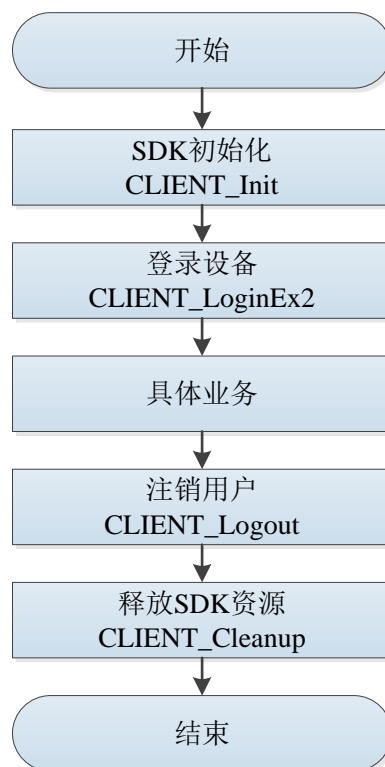
表2-2 设备登录接口信息

接口	说明
CLIENT_LoginEx2	登录扩展接口 2
CLIENT_Logout	登出接口

2.2.3 流程说明

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 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 调用 CLIENT_LoginEx2 登录设备。
- 步骤3 登录成功后，用户可以实现需要的业务功能。
- 步骤4 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤5 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- 登录句柄：登录成功时接口返回值非 0（即句柄可能小于 0，也属于登录成功）；同一设备登录多次，每次的登录句柄不一样。如果无特殊业务，建议只登录一次，登录的句柄可以重复用于其他各种业务。
- 句柄重复：登录句柄有可能与存在过的句柄相同，属于正常现象。例如登录设备 A 获得 loginIDA，将 loginIDA 注销，再次进行登录操作，可能又获取到 LoginIDA。但是在句柄的整个生命周期内，不会出现重复的句柄。
- 登出：接口内部会释放登录会话中已打开的业务，但建议用户不要依赖登出接口的清理功能。例如打开监视后，在不需要使用监视时，用户应该调用结束监视的接口。
- 登录与登出配对使用，登录会消耗一定的内存和 sokcet 信息，在登出后释放资源。
- 登录失败：建议通过登录接口的 error 参数（登录错误码）初步排查。常见错误码请参见表 2-3。

表2-3 常见错误码

error 的错误码	对应的含义
1	密码不正确
2	用户名不存在

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

更多错误码信息请参见《网络 SDK 开发手册.chm》中的“CLIENT_LoginEx2 接口”描述。其中，如果网络状况较差，容易出现错误码 3 时，可以使用以下代码增大超时时间：

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

2.2.4 示例代码

```
NET_DEVICEINFO_Ex stDevInfo = {0};
int nError = 0;
// 登录设备
LLONG lLoginHandle = CLIENT_LoginEx2(szDevIp, nPort,
szUserName,szPasswd,EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfo, &nError);
// 退出设备
if (0 != lLoginHandle)
{
    CLIENT_Logout(lLoginHandle);
}
```

2.3 实时监控

2.3.1 简介

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

SDK 登录设备后，可向设备获取主码流和辅码流。

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

2.3.2 接口总览

表2-4 实时监视接口信息

接口	说明
CLIENT_RealPlayEx	开始实时监视扩展接口
CLIENT_StopRealPlayEx	停止实时监视扩展接口
CLIENT_SaveRealData	开始本地保存实时监视数据
CLIENT_StopSaveRealData	停止本地保存实时监视数据
CLIENT_SetRealDataCallBackEx	设置实时监视数据回调函数扩展接口

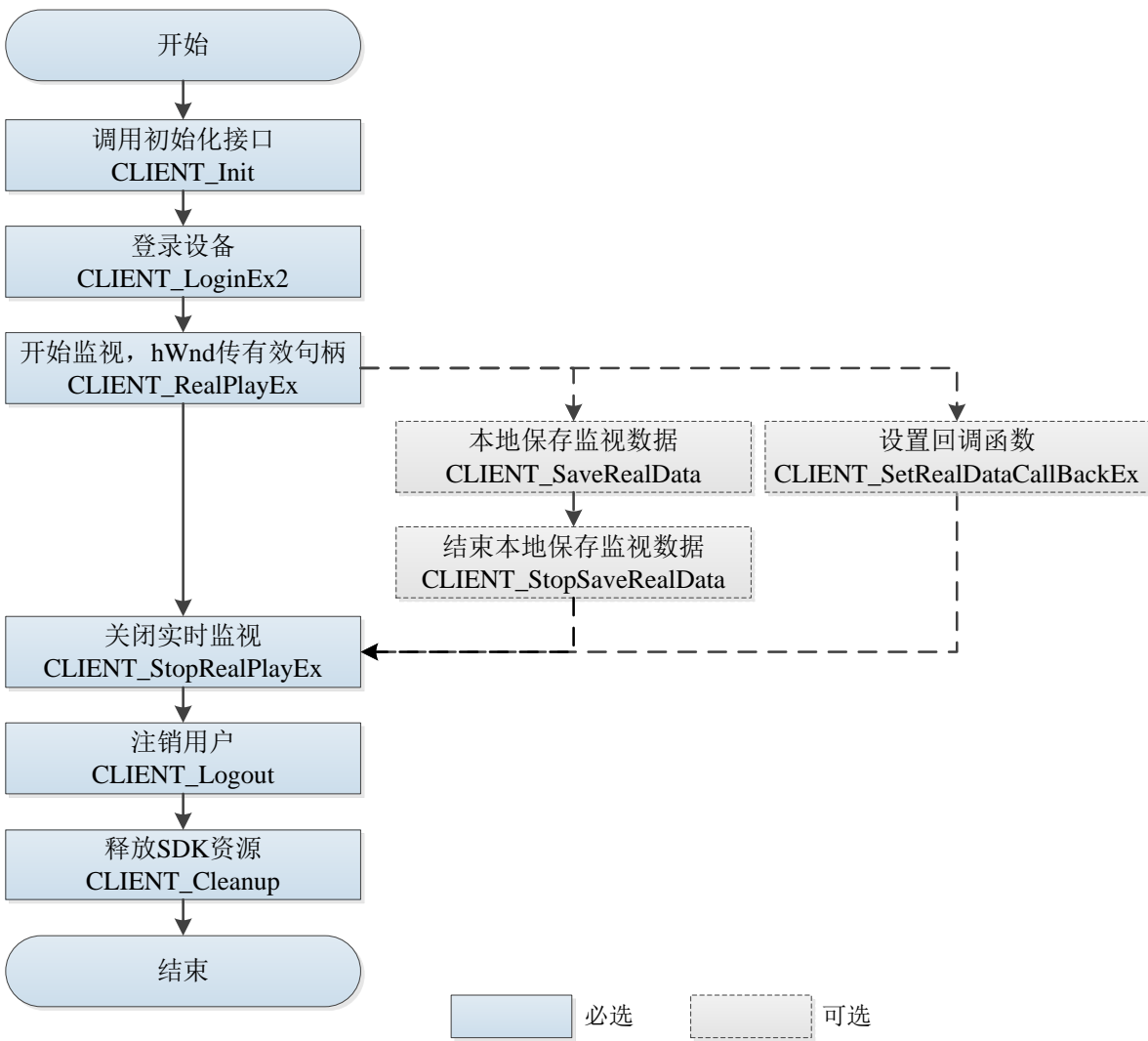
2.3.3 流程说明

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

2.3.3.1 SDK 集成播放库播放

SDK 内部调用辅助库里的 PlaySDK 库实现实时播放。SDK 集成播放库解码播放流程，如图 2-3 所示。

图2-3 SDK 解码播放流程图



流程说明

- 步骤1 调用 `CLIENT_Init` 完成 SDK 初始化流程。
- 步骤2 调用 `CLIENT_LoginEx2` 登录设备。
- 步骤3 调用 `CLIENT_RealPlayEx` 启动实时监视，参数 `hWnd` 为有效窗口句柄。
- 步骤4 （可选）调用 `CLIENT_SaveRealData` 开始保存监视数据。
- 步骤5 （可选）调用 `CLIENT_StopSaveRealData` 结束保存，生成本地视频文件。
- 步骤6 （可选）若调用 `CLIENT_SetRealDataCallBackEx`，用户可将视频数据选择保存或转发。
若保存成文件，与步骤 4、5 效果相同。
- 步骤7 实时监视使用完毕后，调用 `CLIENT_StopRealPlayEx` 停止实时监视。
- 步骤8 业务使用完后，调用 `CLIENT_Logout` 登出设备。
- 步骤9 SDK 功能使用完后，调用 `CLIENT_Cleanup` 释放 SDK 资源。

注意事项

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

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

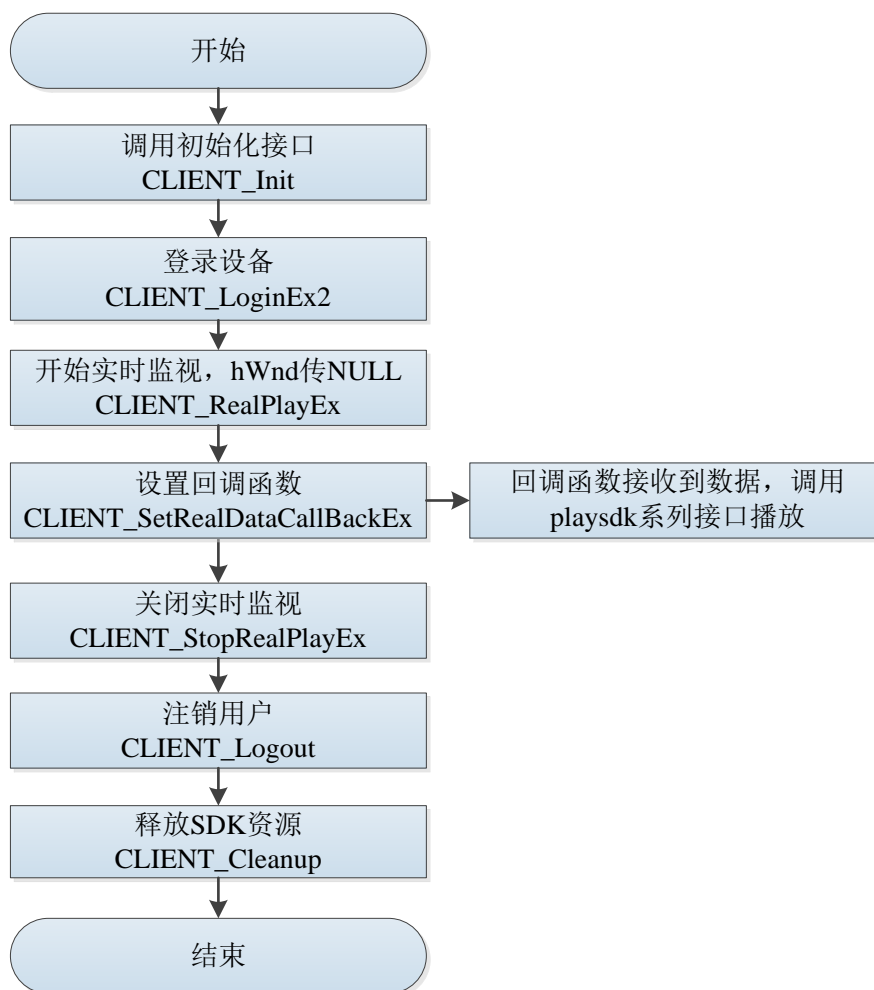
- 重复打开失败：部分设备不支持同一个通道多次打开，当重复打开同一通道的监视，可能会出现第一次打开成功，后续打开失败的现象。建议：
 - ◇ 将已打开的通道先关闭。例如已经开启通道一的主码流视频，希望再打开通道一的辅码流视频时，可先关闭通道一的主码流视频，再开启通道一的辅码流视频。
 - ◇ 登录两次设备获取两个登录句柄，分别处理主码流和辅码流业务。
- 接口成功无画面：SDK 内部解码需要使用到 `dhplay.dll`，建议查看运行目录下是否缺少 `dhplay.dll` 及其依赖的辅助库，具体请参见表 1-1。
- 系统资源不足的情况下，设备可能返回错误而不恢复码流，可以在报警回调函数（即 `CLIENT_SetDVRMessCallBack` 中设置的回调函数）收到事件 `DH_REALPLAY_FAILED_EVENT`，该事件包含了详细的错误码，请参见《网络 SDK 开发手册》中的“`DEV_PLAY_RESULT` 结构体”。
- 32 路限制：解码显示比较消耗资源，特别是高分辨率视频，考虑到客户端硬件资源有限，一般同时解码显示的通道数有限，所以该方式暂时限定为最多 32 路，如超过 32 路，建议使用“2.3.3.2 调用私有播放库播放”。

2.3.3.2 调用私有播放库播放

SDK 回调实时监视码流给用户，用户调用 `PlaySDK` 进行解码播放。用户调用私有播放库解码播

放流程如图 2-4 所示。

图2-4 第三方解码播放流程图



流程说明

- 步骤1 调用 `CLIENT_Init` 完成 SDK 初始化流程。
- 步骤2 调用 `CLIENT_LoginEx2` 登录设备。
- 步骤3 登录成功后, 调用 `CLIENT_RealPlayEx` 启动实时监控, 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 在回调函数中将数据传给 PlaySDK 完成解码。
- 步骤6 实时监控使用完毕后, 调用 `CLIENT_StopRealPlayEx` 停止实时监控。
- 步骤7 业务使用完后, 调用 `CLIENT_Logout` 登出设备。
- 步骤8 SDK 功能使用完后, 调用 `CLIENT_Cleanup` 释放 SDK 资源。

注意事项

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

2.3.4 示例代码

2.3.4.1 SDK 集成播放库播放

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

2.3.4.2 调用私有播放库播放

```
// 以开启第一路的主码流监视为例
LLONG IRealHandle = CLIENT_RealPlayEx(ILoginHandle, 0, NULL, DH_RType_Realplay);
if (NULL == IRealHandle)
{
    printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
else
{
    DWORD dwFlag = 0x00000001;
    CLIENT_SetRealDataCallBackEx(IRealHandle, &RealDataCallBackEx, NULL, dwFlag);
}
// 关闭预览
if (0 != IRealHandle)
{
    CLIENT_StopRealPlayEx(IRealHandle);
}

void CALLBACK RealDataCallBackEx(LLONG IRealHandle, DWORD dwDataType, BYTE *pBuffer,
DWORD dwBufSize, LONG param, LDWORD dwUser)
{

```

```
// 从设备获取的码流数据，需调用 PlaySDK 的接口，详见 SDK 监视 demo 源码
printf("receive real data, param: IRealHandle[%p], dwDataType[%d], pBuffer[%p], dwBufSize[%d]\n",
IRealHandle, dwDataType, pBuffer, dwBufSize);
}
```

2.4 智能事件订阅

2.4.1 简介

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

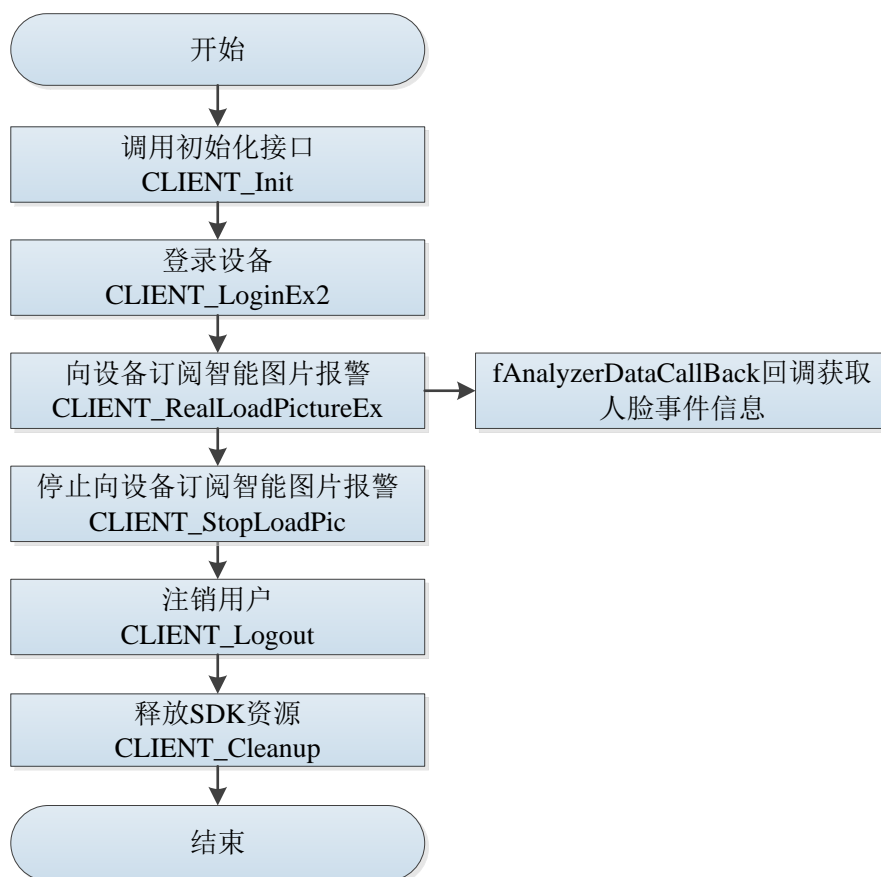
2.4.2 接口总览

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

2.4.3 流程说明

人脸事件上报流程，如图 2-5 所示。

图2-5 人脸事件上报流程



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用 CLIENT_RealLoadPictureEx 向设备订阅智能事件。
- 步骤4 订阅成功后，设备上报的智能事件通过 fAnalyzerDataCallBack 回调函数上报智能事件。通过该函数，可以根据报警类型过滤出您需要的智能报警事件。
- 步骤5 智能事件上报功能使用完毕后，调用 CLIENT_StopLoadPic 停止订阅智能事件。
- 步骤6 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤7 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- 订阅事件类型：如果需要同时上报不同智能事件时，支持订阅所有智能事件（EVENT_IVS_ALL）；也支持订阅单个智能事件。
- 设置接收图片缓存：由于 SDK 默认接收缓存是 2M，当回调上来的图片数据大于 2M 时，用户需调用 CLIENT_SetNetworkParam 接口设置接收图片缓存，否则将丢弃大于 2M 的数据包。
- 设置是否接收图片：由于一些设备所在网络环境是 3G 或 4G 网络，当 SDK 连接设备时，如不需要接收图片可以把 CLIENT_RealLoadPictureEx 接口中 bNeedPicFile 参数设置为 False，只接收人脸事件信息，不带图片。

2.4.4 示例代码

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

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

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

2.5 查询/回放/下载录像和图片

2.5.1 简介

设备的智能算法在分析实时码流时，如果检测为某个智能事件，就会触发对该智能事件的录像和抓拍，并将其保存。用户可以查询设备中保存下来的智能事件的录像和图片，并对查询到的结果进行下载或者回放操作。

2.5.2 接口总览

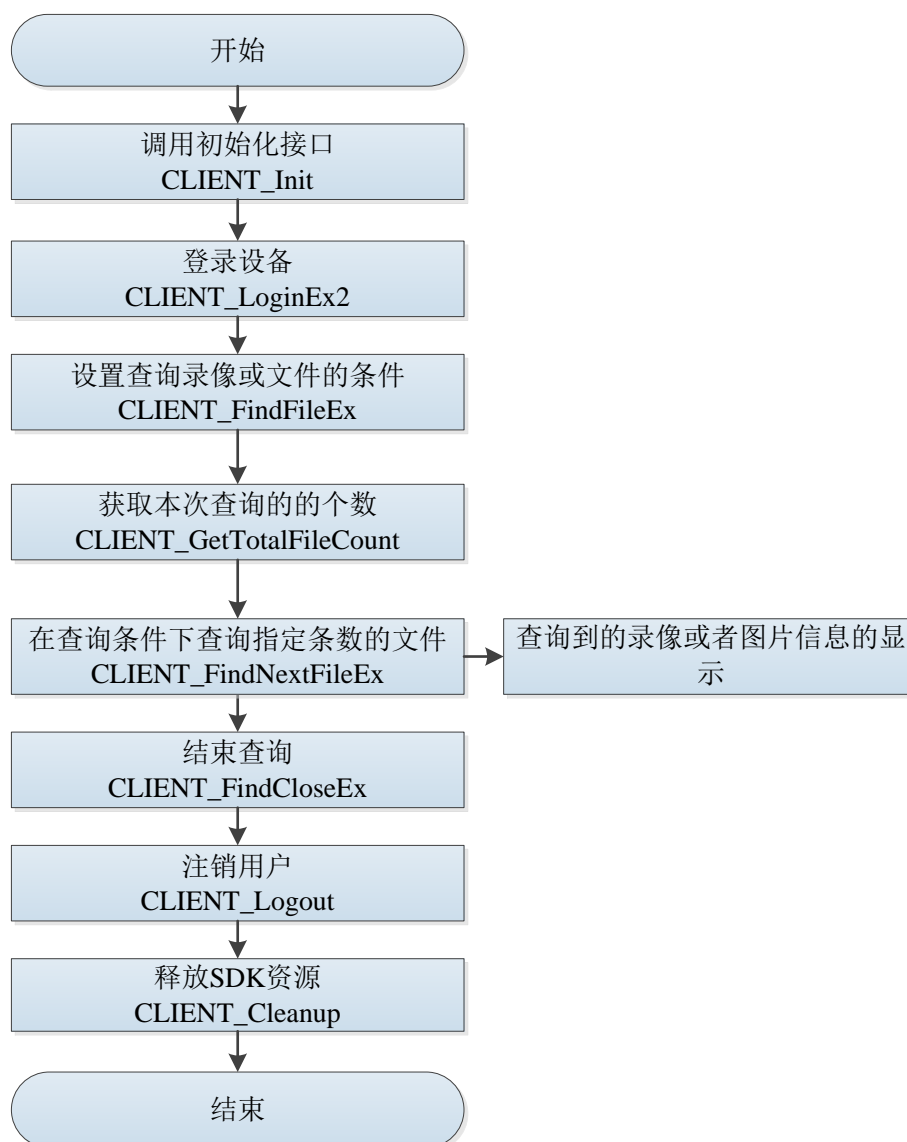
接口	说明
CLIENT_FindFileEx	按条件查询录像或者图片，设置查询条件
CLIENT_GetTotalFileCount	获取本次查询的录像或者图片的个数
CLIENT_FindNextFileEx	查询指定的条数的录像或者图片
CLIENT_FindCloseEx	结束查询
CLIENT_PlayBackByTimeEx2	按时间开始录像的回放
CLIENT_StopPlayBack	停止录像回放
CLIENT_DownloadByTimeEx	录像下载
CLIENT_StopDownload	停止录像下载
CLIENT_DownloadRemoteFile	下载图片

2.5.3 流程说明

2.5.3.1 录像或图片的查询流程

录像或图片的查询流程，如图 2-6 所示。

图2-6 录像或图片的查询流程



流程说明

- 步骤1 调用 `CLIENT_Init` 完成 SDK 初始化流程。
- 步骤2 调用 `CLIENT_LoginEx2` 登录设备。
- 步骤3 调用 `CLIENT_FindFileEx` 设置查询条件，设置成功后返回查询句柄。根据 `emType` 的不同取值，判断查找的类型。
- 步骤4 调用 `CLIENT_GetTotalFileCount` 获取查询到的录像或者文件总数。
- 步骤5 调用 `CLIENT_FindNextFileEx` 查询指定条数的录像或者图片，并将查询到的信息保存，用于录像或者图片的回放下载操作。
- 步骤6 调用 `CLIENT_FindCloseEx` 结束查询。
- 步骤7 业务使用完后，调用 `CLIENT_Logout` 登出设备。
- 步骤8 SDK 功能使用完后，调用 `CLIENT_Cleanup` 释放 SDK 资源。

注意事项

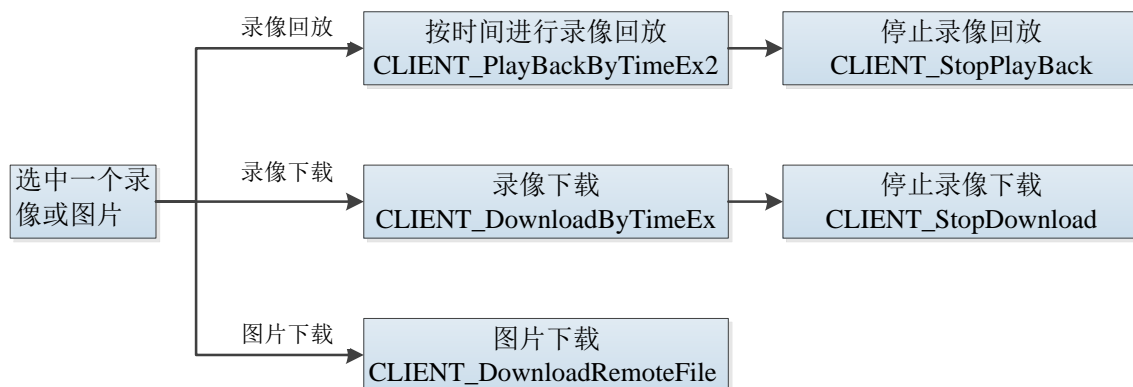
- 接口 `CLIENT_FindFileEx` 中参数 `pQueryCondition` 是用户申请和释放的，具体类型由 `emType` 的枚举类型来定义。

- **CLIENT_FindFileEx** 如果查询成功，则会返回查询句柄，**CLIENT_FindNextFileEx** 将查询句柄作为参数查询具体的录像或者图片，且必须调用 **CLIENT_FindCloseEx** 将查询句柄关闭。
- **CLIENT_FindNextFileEx** 可设置查询条数，如果设置条数大于 1 条则参数 **pMediaFileInfo** 必须为一个数据指针，数据大小大于等于设置的条数。

2.5.3.2 录像的回放和下载、图片的下载流程

录像的回放和下载、图片的下载流程，如图 2-7 所示。

图2-7 录像的回放和下载、图片的下载流程



流程说明

选择一条接口 **CLIENT_FindNextFileEx** 查询到的结果信息，然后选择下载还是回放。

- 录像回放

步骤1 如果是录像文件，使用录像查询结果中的开始时间和结束时间，调用接口 **CLIENT_PlayBackByTimeEx2** 进行回放。

步骤2 回放过程中或者回放结束后，调用接口 **CLIENT_StopPlayBack** 进行录像回放的停止。

- 录像下载

步骤1 如果是录像文件，使用录像查询结果中的开始时间和结束时间，调用接口 **CLIENT_DownloadByTimeEx** 进行录像的下载。

步骤2 下载结束后，调用 **CLIENT_StopDownload** 接口进行录像下载的停止。

- 图片下载

如果选择的是图片文件，使用图片查询结果中的文件名和图片类型，调用接口 **CLIENT_DownloadRemoteFile** 下载图片。

注意事项

录像的回放与下载，以及图片的下载都需要依赖录像和图片的查询结果，并以查询到信息作为回放和下载的条件。

2.5.4 示例代码

2.5.4.1 录像或图片的查询

```
// 查询条件
```



```

MEDIAFILE_FACE_DETECTION_PARAM param;
memset(&param, 0, sizeof(param));
param.dwSize = sizeof(param);
param.stuDetail.dwSize = sizeof(MEDIAFILE_FACE_DETECTION_DETAIL_PARAM);
param.nChannelID = -1;
param.stuStartTime = startTime;
param.stuEndTime = endTime
param.emPicType = NET_FACEPIC_TYPE_SMALL; // 人脸小图
param.bDetailEnable = FALSE;
param.emSex = EM_DEV_EVENT_FACEDETECT_SEX_TYPE_MAN;
param.bAgeEnable = FALSE;
param.nEmotionValidNum = 0;
param.emGlasses = EM_FACEDETECT_WITH_GLASSES;

// 查询人脸检测的小图
LLONG lFindFileHandle = CLIENT_FindFileEx(g_lLoginHandle, DH_FILE_QUERY_FACE_DETECTION,
&param, NULL, 5000);
if (lFindFileHandle == 0)
{
    printf("CLIENT_FindFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
    return;
}

// 获取查询到的人脸个数
BOOL nRet = CLIENT_GetTotalFileCount(lFindFileHandle, &nCount, NULL);
if (!nRet)
{
    printf("CLIENT_GetTotalFileCount: failed! Error code: %x.\n", CLIENT_GetLastError());
    return;
}

// 查询个数
int nMaxConut = 10;
MEDIAFILE_FACE_DETECTION_INFO* pMediaFileInfo = NEW
MEDIAFILE_FACE_DETECTION_INFO[nMaxConut];

// 开始查询
int nRet = CLIENT_FindNextFileEx(lFindFileHandle, nMaxConut, (void*)pMediaFileInfo, nMaxConut *
sizeof(MEDIAFILE_FACE_DETECTION_INFO), NULL, 3000);
if (nRet < 0)

```

```

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

```

// 关闭查询

```
CLIENT_FindCloseEx(IFindFileHandle);
```

2.5.4.2 录像回放

// 设置回放时的码流类型，此处设置成主码流

```
int nStreamType = 0; // 0-主辅码流， 1-主码流， 2-辅码流
```

```
CLIENT_SetDeviceMode(ILoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);
```

// 设置回放时的录像文件类型，此处设置成所有录像

```
NET_RECORD_TYPE emFileType = NET_RECORD_TYPE_ALL; // 所有录像
```

```
CLIENT_SetDeviceMode(ILoginHandle, DH_RECORD_TYPE, &emFileType);
```

// 开启录像回放

```
int nChannelID = 0; // 通道号
```

```
NET_IN_PLAY_BACK_BY_TIME_INFO stIn = {0};
```

```
NET_OUT_PLAY_BACK_BY_TIME_INFO stOut = {0};
```

```
memcpy(&stIn.stStartTime, &stuStartTime, sizeof(stuStartTime));
```

```
memcpy(&stIn.stStopTime, &stuStopTime, sizeof(stuStopTime));
```

```
stIn.hWnd = hWnd;
```

```
stIn.fDownloadDataCallBack = DataCallBack;
```

```
stIn.dwDataUser = NULL;
```

```
stIn.cbDownloadPos = NULL;
```

```
stIn.dwPosUser = NULL;
```

```
stIn.nPlayDirection = emDirection;
```

```
stIn.nWaittime = 10000;
```

```
LLONG lPlayHandle = CLIENT_PlayBackByTimeEx2(ILoginHandle, nChannelID, &stIn, &stOut);
```

```
if (0 == lPlayHandle)
```

```

{
    printf("CLIENT_PlayBackByTimeEx2: failed! Error code: %x.\n", CLIENT_GetLastError());
}

```

```
if (FALSE == CLIENT_StopPlayBack(lPlayHandle))
```

```

{
    printf("CLIENT_StopPlayBack Failed, lRealHandle[%x]!Last Error[%x]\n", lPlayHandle,
        CLIENT_GetLastError());
}

```

2.5.4.3 录像下载

```
//回放进度函数
void CALLBACK TimeDownLoadPosCallBack(LLONG IPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser);

// 回放/下载数据回调函数
int CALLBACK DataCallBack(LLONG IRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
dwBufSize, LDWORD dwUser);

int main()
{
    // 设置查询时的录像码流类型，此处设置码流类型为主辅码流
    int nStreamType = 0; // 0-主辅码流， 1-主码流， 2-辅码流
    CLIENT_SetDeviceMode(ILoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

    // 设置下载开始和结束时间
    int nChannelID = 0; // 通道号

    NET_TIME stuStartTime = {0};
    stuStartTime.dwYear = 2018;
    stuStartTime.dwMonth = 9;
    stuStartTime.dwDay = 17;

    NET_TIME stuStopTime = {0};
    stuStopTime.dwYear = 2018;
    stuStopTime.dwMonth = 9;
    stuStopTime.dwDay = 18;

    // 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
    IDownloadHandle = CLIENT_DownloadByTimeEx(ILoginHandle, nChannelID,
    EM_RECORD_TYPE_ALL, &stuStartTime, &stuStopTime, "test.dav", TimeDownLoadPosCallBack, NULL,
    DataCallBack, NULL);

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

    // 关闭下载，可在下载结束后调用，也可在下载中调用
```

```

    if (0 != IDownloadHandle)
    {
        if (!CLIENT_StopDownload(IDownloadHandle))
        {
            printf("CLIENT_StopDownload Failed, IDownloadHandle[%x]!Last Error[%x]\n" ,
                IDownloadHandle, CLIENT_GetLastError());
        }
    }
}

void CALLBACK TimeDownLoadPosCallBack(LLONG IPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser)
{
    // 用户对进度回调做处理
}

int CALLBACK DataCallBack(LLONG IRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
dwBufSize, LDWORD dwUser)
{
    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
        break;
    case 1://Standard video data
        break;
    case 2: //yuv data
        break;
    case 3://pcm audio data
        break;
    default:
        break;
    }
    return 0;
}

```

2.5.4.4 图片下载

```
DH_IN_DOWNLOAD_REMOTE_FILE stuRemoteFileParm;
```

```
memset(&stuRemoteFileParm, 0, sizeof(DH_IN_DOWNLOAD_REMOTE_FILE));
stuRemoteFileParm.dwSize = sizeof(DH_IN_DOWNLOAD_REMOTE_FILE);
stuRemoteFileParm.pszFileName = pInfo->stObjectPic.szFilePath ;
stuRemoteFileParm.pszFileDst = szFileName;

DH_OUT_DOWNLOAD_REMOTE_FILE *fileinfo = NEW DH_OUT_DOWNLOAD_REMOTE_FILE;
fileinfo->dwSize = sizeof(DH_OUT_DOWNLOAD_REMOTE_FILE);

if (!CLIENT_DownloadRemoteFile(g_hLoginHandle, &stuRemoteFileParm, fileinfo))
{
    printf("CLIENT_DownloadRemoteFile Failed,Last Error[%x]\n" , CLIENT_GetLastError());
}
```

3.1 人脸事件订阅

具体的订阅请参见“2.4 智能事件订阅”，在智能事件上报的回调函数 `fAnalyzerDataCallBack` 中过滤出人脸检测与人脸识别事件，即 `EVENT_IVS_FACEDetect` 人脸检测事件和 `EVENT_IVS_FACERecognition` 人脸识别事件。

3.2 人脸库的增删改查

3.2.1 简介

人脸库即人脸图片和人员信息的一个集合，提供了人脸库的添加、删除、修改和查找功能。

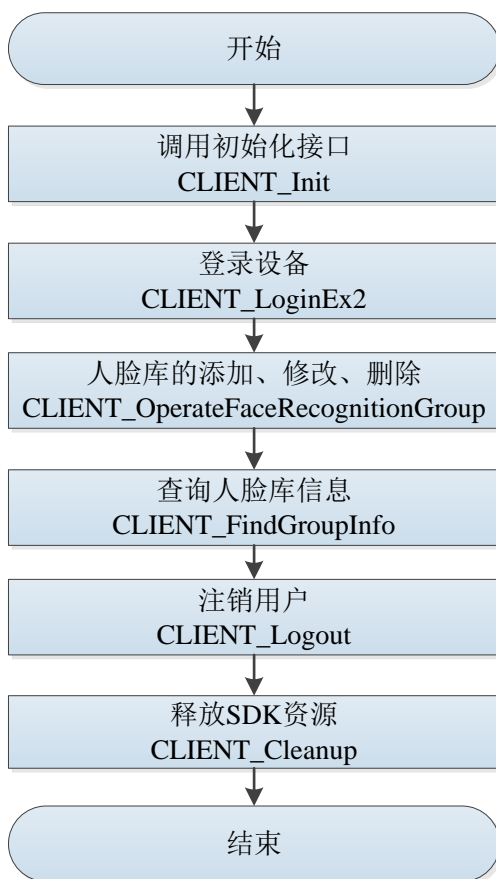
3.2.2 接口总览

接口	说明
<code>CLIENT_OperateFaceRecognitionGroup</code>	人脸库的添加、修改和删除操作
<code>CLIENT_FindGroupInfo</code>	人脸库信息的查询

3.2.3 流程说明

人脸库操作流程，如图 3-1 所示。

图3-1 人脸库操作流程



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用 CLIENT_OperateFaceRecognitionGroup 根据相应的枚举类型进行人脸库的添加、修改、删除操作。
- 步骤4 调用 CLIENT_FindGroupInfo 获取人脸库信息。
- 步骤5 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤6 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- 人脸库添加：添加操作时，操作类型 emOperateType 对应的值取 NET_FACERECONGNITION_GROUP_ADD，所对应的结构体为 NET_ADD_FACERECONGNITION_GROUP_INFO。
- 人脸库修改：修改人脸库操作类型 emOperateType 对应值为 NET_FACERECONGNITION_GROUP_MODIFY，对应结构体为 NET_MODIFY_FACERECONGNITION_GROUP_INFO；修改人脸库需要指定 GroupID 和人脸库类型 emFaceDBType，所指定的 GroupID 需要设备上已存在的。
- 人脸库删除：删除人脸库操作类型 emOperateType 对应值为 NET_FACERECONGNITION_GROUP_DELETE，对应结构体为 NET_DELETE_FACERECONGNITION_GROUP_INFO；删除人脸库时若指定了 GroupID，则删除对应 GroupID 的人脸库，若未指定 GroupID，则删除全部的人脸库。

3.2.4 示例代码

3.2.4.1 人脸库信息的查询

```
// 设置人脸库查询条件
NET_IN_FIND_GROUP_INFO stuInParam = {sizeof(stuInParam)};
NET_OUT_FIND_GROUP_INFO stuOutParam = {sizeof(stuOutParam)};
stuOutParam.nMaxGroupNum = 100;
NET_FACERECONGNITION_GROUP_INFO *pGroupInfo = NULL;
stuOutParam.pGroupInfos = new NET_FACERECONGNITION_GROUP_INFO[100];
memset(stuOutParam.pGroupInfos, 0, sizeof(NET_FACERECONGNITION_GROUP_INFO)*100);

// 人脸库查询
BOOL bRet = CLIENT_FindGroupInfo(lLoginHandle, &stuInParam, &stuOutParam, 5000);
if(FALSE == bRet)
{
    printf("CLIENT_FindGroupInfo: failed! Error code %x.\n", CLIENT_GetLastError());
    return -1;
}
delete[] pGroupInfo;
```

3.2.4.2 人脸库的增删改

```
enum EM_OPERATION_TYPE
{
    FACEDB_DELETE, // 删除
    FACEDB_ADD,    // 添加
    FACEDB_MODIFY  // 修改
};

// 设置需要删除的人脸库 ID
NET_FACERECONGNITION_GROUP_INFO *pstGroupInfo = m_pstSelectGroup;
NET_IN_OPERATE_FACERECONGNITION_GROUP stuInParam = {sizeof(stuInParam)};
NET_OUT_OPERATE_FACERECONGNITION_GROUP stuOutParam = {sizeof(stuOutParam)};
NET_ADD_FACERECONGNITION_GROUP_INFO stuAddGroupInfo = {sizeof(stuAddGroupInfo)};
NET_MODIFY_FACERECONGNITION_GROUP_INFO stuEditGroupInfo = {sizeof(stuEditGroupInfo)};

EM_OPERATION_TYPE emType = mType;
switch(emType)
{
    // 删除人脸库
```



```

case FACEDB_DELETE:
{
    stuInParam.emOperateType = NET_FACERECONGNITION_GROUP_DELETE;
    NET_DELETE_FACERECONGNITION_GROUP_INFO stuDeleteInfo;
    memset(&stuDeleteInfo, 0, sizeof(stuDeleteInfo));
    stuDeleteInfo.dwSize = {sizeof(stuDeleteInfo)};
    strncpy(stuDeleteInfo.szGroupId, pstGroupInfo->szGroupId, sizeof(stuDeleteInfo.szGroupId)-1);
    stuInParam.pOperateInfo = &stuDeleteInfo;
    break;
}
// 添加人脸库
case FACEDB_ADD:
{
    stuInParam.emOperateType = NET_FACERECONGNITION_GROUP_ADD;
    stuAddGroupInfo.stuGroupInfo.dwSize = sizeof(stuAddGroupInfo.stuGroupInfo);
    stuAddGroupInfo.stuGroupInfo.emFaceDBType = NET_FACE_DB_TYPE_BLACKLIST;

    strncpy(stuAddGroupInfo.stuGroupInfo.szGroupName, pcGroupName, sizeof(stuAddGroupInfo.stuGroup
pInfo.szGroupName)-1);
    stuInParam.pOperateInfo = &stuAddGroupInfo;
    break;
}
// 修改人脸库
case FACEDB_MODIFY:
{
    stuInParam.emOperateType = NET_FACERECONGNITION_GROUP_MODIFY;
    stuEditGroupInfo.stuGroupInfo.dwSize = sizeof(stuEditGroupInfo.stuGroupInfo);
    stuEditGroupInfo.stuGroupInfo.emFaceDBType = NET_FACE_DB_TYPE_BLACKLIST;
    strncpy(stuEditGroupInfo.stuGroupInfo.szGroupName, pcGroupName,
sizeof(stuEditGroupInfo.stuGroupInfo.szGroupName)-1);
    strncpy(stuEditGroupInfo.stuGroupInfo.szGroupId, m_stuGroupInfo.szGroupId,
sizeof(stuEditGroupInfo.stuGroupInfo.szGroupId)-1);
    stuInParam.pOperateInfo = &stuEditGroupInfo;
    break;
}
default:
    break;
}
}
BOOL bRet = CLIENT_OperateFaceRecognitionGroup(m_loginID, &stuInParam, &stuOutParam, 5000);
if(FALSE == bRet)

```

```
{  
    printf("CLIENT_ OperateFaceRecognitionGroup: failed! Error code %x.\n", CLIENT_GetLastError());  
    return -1;  
}
```

3.3 人脸的增删改查

3.3.1 简介

人脸库中存放着相应的人员即人脸信息，通过调用该功能接口可以完成人脸信息的添加、修改、查找、删除操作。

3.3.2 接口总览

接口	说明
CLIENT_OperateFaceRecognitionDB	人脸的添加、修改和删除操作
CLIENT_StartFindFaceRecognition	设置人脸的查询条件
CLIENT_DoFindFaceRecognition	查找指定条数的人脸数据
CLIENT_StopFindFaceRecognition	结束查询

3.3.3 流程说明

人脸的增删改查流程，如图 3-2 所示。

图3-2 人脸的增删改查流程



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用 CLIENT_OperateFaceRecognitionDB 根据相应的枚举类型进行人脸库的添加、修改、删除操作。
- 步骤4 调用 CLIENT_StartFindFaceRecognition 设置人脸查询条件。
- 步骤5 调用 CLIENT_DoFindFaceRecognition 接口获取查询结果。
- 步骤6 调用 CLIENT_StopFindFaceRecognition 接口结束查询。
- 步骤7 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤8 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- 人脸库人脸添加：添加操作时，操作类型 emOperateType 对应的值取 NET_FACERECONGNITIONDB_ADD，入参结构体中 pBuffer 保存需要添加的人脸图片，用户自己申请和释放内存。GroupID 必填，其他按需填写。若添加人员成功，设备会返回 UID，即人员唯一标识符，用于表示唯一人员，见 pstOutParam -> szUID。
- 人脸库人脸修改：修改人脸操作类型 emOperateType 对应值为

NET_FACERECONGNITIONDB_MODIFY，入参结构体中 pBuffer 保存需要置换的人脸图片（设备上原有的样本图片删除，替换成现在新下发的图片），用户自己申请和释放内存；stPersonInfo 需要修改的人员信息，GroupID（可通过“查询当前人脸库信息”得到）和 szUID（人员唯一标识符，可通过“查找人员信息”）必须填写，其他按需填写。

- 人脸库删除：删除人脸库操作类型 emOperateType 对应值为 NET_FACERECONGNITIONDB_DELETE；stPersonInfo 需要删除的人员信息，GroupID（可通过“查询当前人脸库信息”得到）和 szUID（人员唯一标识符，可通过“查找人员信息”）。

3.3.4 示例代码

3.3.4.1 人脸的增删改

```
// 人员的增加与修改
NET_IN_OPERATE_FACERECONGNITIONDB stuInParam = { sizeof(stuInParam) };
NET_OUT_OPERATE_FACERECONGNITIONDB stuOutParam = { sizeof(stuOutParam) };
// 添加人员信息
{
    stuInParam.emOperateType = NET_FACERECONGNITIONDB_ADD;
}

// 修改人员信息
{
    //stuInParam.emOperateType = NET_FACERECONGNITIONDB_MODIFY;
    //strncpy(stuInParam.stPersonInfoEx.szUID, strUID, sizeof(stuInParam.stPersonInfoEx.szUID) - 1);
}

stuInParam.bUsePersonInfoEx = TRUE;
stuInParam.stPersonInfoEx.bySex = 1; // 男
stuInParam.stPersonInfoEx.byIDType = 1; // 身份证
stuInParam.stPersonInfoEx.wYear = time.GetYear();
stuInParam.stPersonInfoEx.byMonth = time.GetMonth();
stuInParam.stPersonInfoEx.byDay = time.GetDay();
strncpy(stuInParam.stPersonInfoEx.szPersonName, pstrName, sizeof(stuInParam.stPersonInfoEx.szPersonName) - 1);
strncpy(stuInParam.stPersonInfoEx.szID, pstrCardID, sizeof(stuInParam.stPersonInfoEx.szID) - 1);
strncpy(stuInParam.stPersonInfoEx.szGroupName, m_szGroupName,
sizeof(stuInParam.stPersonInfoEx.szGroupName) - 1);
strncpy(stuInParam.stPersonInfoEx.szGroupID, m_szGroupId, sizeof(stuInParam.stPersonInfoEx.szGroupID) - 1);
stuInParam.nBufferLen = nPictureBufferLen;
stuInParam.pBuffer = pPictureBuffer;
stuInParam.stPersonInfoEx.wFacePicNum = 1;
stuInParam.stPersonInfoEx.szFacePicInfo[0].dwOffSet = 0;
stuInParam.stPersonInfoEx.szFacePicInfo[0].dwFileLenth = nLength;

bRet = CLIENT_OperateFaceRecognitionDB(m_ILoginID, &stuInParam, &stuOutParam, 5000);
```

```

if (FACE_PERSON_ADD == m_nOpreateType)
{
    printf("CLIENT_OperateFaceRecognitionDB failed! Error code %x.\n", CLIENT_GetLastError());
    return -1;
}

// 删除人员信息
NET_IN_OPERATE_FACERECONGNITIONDB stuInParam = { sizeof(stuInParam) };
NET_OUT_OPERATE_FACERECONGNITIONDB stuOutParam = { sizeof(stuOutParam) };
// 只需要 szGroupID 和 szUID 信息，其他不需要配置
stuInParam.emOperateType = NET_FACERECONGNITIONDB_DELETE;
stuInParam.bUsePersonInfoEx = TRUE;
strncpy(stuInParam.stPersonInfoEx.szUID,
m_pstPersonSelectInfo->stuCandidate.stPersonInfo.szUID, sizeof(stuInParam.stPersonInfoEx.szUID) - 1);
strncpy(stuInParam.stPersonInfoEx.szGroupID, m_szGroupId, sizeof(stuInParam.stPersonInfoEx.szGroupID) - 1);

BOOL bRet = CLIENT_OperateFaceRecognitionDB(ILLoginHandle, &stuInParam, &stuOutParam, 5000);
if (!bRet)
{
    printf("CLIENT_OperateFaceRecognitionDB failed! Error code %x.\n", CLIENT_GetLastError());
    return -1;
}

```

3.3.4.2 人脸的查询

```

// 设置人脸查询条件
NET_IN_STARTFIND_FACERECONGNITION stuInParam = { sizeof(stuInParam) };
NET_OUT_STARTFIND_FACERECONGNITION stuOutParam = { sizeof(stuOutParam) };

stuInParam.stMatchOptions.dwSize = sizeof(stuInParam.stMatchOptions);
stuInParam.stFilterInfo.dwSize = sizeof(stuInParam.stFilterInfo);
stuInParam.bPersonExEnable = TRUE;
stuInParam.stFilterInfo.nRangeNum = 1;
stuInParam.stFilterInfo.szRange[0] = (BYTE)NET_FACE_DB_TYPE_BLACKLIST;
strncpy(stuInParam.stPersonInfoEx.szPersonName, m_PersonName,
sizeof(stuInParam.stPersonInfoEx.szPersonName)-1);
stuInParam.stPersonInfoEx.bySex = 0;
stuInParam.stFilterInfo.stBirthdayRangeStart = BirthdayRangeStart;
stuInParam.stFilterInfo.stBirthdayRangeEnd = BirthdayRangeEnd;
strncpy(stuInParam.stPersonInfoEx.szID, pcCard, sizeof(stuInParam.stPersonInfoEx.szID)-1);
strncpy(stuInParam.stFilterInfo.szGroupId[0], m_szGroupId, sizeof(m_szGroupId)-1);
stuInParam.stFilterInfo.nGroupIdNum = 1;

```

```

strncpy(stuInParam.stPersonInfoEx.szGroupID, m_szGroupId, sizeof(stuInParam.stPersonInfoEx.szGroupID)-1);

BOOL bRet = CLIENT_StartFindFaceRecognition(m_lLoginID, &stuInParam, &stuOutParam,
DEFAULT_WAIT_TIME);
if (!bRet)
{
    printf("CLIENT_StartFindFaceRecognition failed! Error code %x.\n", CLIENT_GetLastError());
    return;
}

// 开始进行查询
NET_IN_DOFIND_FACERECONGNITION stuInDoFind = {sizeof(stuInDoFind)};
NET_OUT_DOFIND_FACERECONGNITION stuOutDoFind = {sizeof(stuOutDoFind)};
stuOutDoFind.bUseCandidatesEx = TRUE;

stuInDoFind.lFindHandle = m_lFindPersonHandle;
stuInDoFind.emDataType = EM_NEEDED_PIC_TYPE_HTTP_URL;
stuInDoFind.nCount = 10;
stuInDoFind.nBeginNum = m_nCurPos;
bRet = CLIENT_DoFindFaceRecognition(&stuInDoFind, &stuOutDoFind, WAIT_TIMEOUT);
if (!bRet)
{
    printf("CLIENT_DoFindFaceRecognition failed! Error code %x.\n", CLIENT_GetLastError());
    return;
}

```

3.4 以通道或者库为对象布控

3.4.1 简介

以通道为对象进行布控，即一个通道可布控一个或者多个人脸库。

以库为对象进行布控，即一个人脸库可布控一个或者多个通道。

两种方式都为人脸库的布控。

3.4.2 接口总览

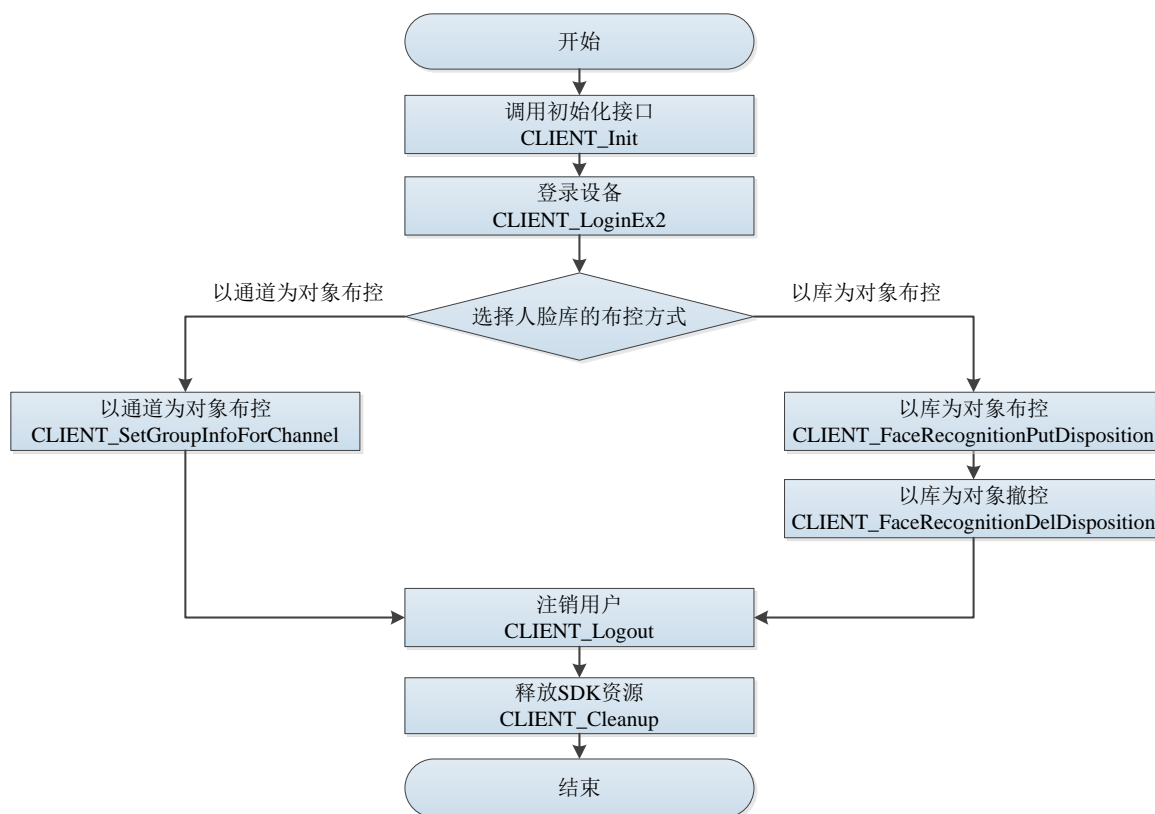
接口	说明
CLIENT_FaceRecognitionPutDisposition	以库为对象进行布控
CLIENT_FaceRecognitionDelDisposition	以库为对象进行撤控

接口	说明
CLIENT_SetGroupInfoForChannel	以通道为对象进行布控

3.4.3 流程说明

以通道或者库为对象进行布控流程，如图 3-3 所示。

图3-3 以通道或者库为对象进行布控流程



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 选择不同的人脸库的布控方式，有以库为对象进行布控，以通道为对象进行布控。
- 以库为对象布控
 1. 选择以库为对象，调用 CLIENT_FaceRecognitionPutDisposition 对库进行布控。
 2. 业务使用完后，调用 CLIENT_FaceRecognitionDelDisposition 对库进行撤控。
 - 以通道为对象布控

选择以通道为对象，调用 CLIENT_SetGroupInfoForChannel 对通道布控。
- 步骤4 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤5 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- 以通道为对象或者以库为对象布控都是人脸库布控的方式。
- 以通道为对象布控时，可将多个人脸库部署到一个通道上。以库为对象布控时，可将一个人脸库部署到多个通道上。
- 以通道为对象布控时，接口 CLIENT_SetGroupInfoForChannel 布控方式为覆盖方式，即总是

以最新的布控配置为主。撤控操作为将空的配置布控下去即可。

- 以库为对象撤控时,接口 `CLIENT_FaceRecognitionDelDisposition` 可对人脸库已经部署的部分通道做撤防,例如:人脸库已经部署在 3 个通道上,撤控的时候可对其部署通道的其中 2 个通道撤控,剩余一个通道保持原样。

3.4.4 示例代码

3.4.4.1 以通道为对象布控

```
// 输入参数
NET_IN_SET_GROUPINFO_FOR_CHANNEL stInChannelDeploy =
{ sizeof(NET_IN_SET_GROUPINFO_FOR_CHANNEL)};
stInChannelDeploy.nChannelID = 0;
stInChannelDeploy.nGroupIdNum = 2; // 需要在该通道配置人脸库的个数
strncpy(stInChannelDeploy.szGroupId[0], strGroupId1, DH_COMMON_STRING_64-1); // 对人脸库 ID 进行
拷贝
strncpy(stInChannelDeploy.szGroupId[1], strGroupId2, DH_COMMON_STRING_64-1);
stInChannelDeploy.nSimilaryNum = 2; // 相似度阈值个数, 与人员组数相同
stInChannelDeploy.nSimilary[0] = 85; // 第一个人脸库的人脸相似度阈值
stInChannelDeploy.nSimilary[1] = 90; // 第二个人脸库的人脸相似度阈值

// 输出参数
NET_OUT_SET_GROUPINFO_FOR_CHANNEL stOutChannelDeploy =
{ sizeof(NET_OUT_SET_GROUPINFO_FOR_CHANNEL)};

// 以库为对象布控
BOOL bRet = CLIENT_SetGroupInfoForChannel(ILoginHandle, &stInChannelDeploy, &stOutChannelDeploy);
if (flase == bRet)
{
    printf("CLIENT_SetGroupInfoForChannel: failed! Error code: %x.\n", CLIENT_GetLastError());
}

// 以通道为对象撤控, 将空的布控信息下发
if (NULL != IRealHandle)
{
    memset(stInChannelDeploy, 0, sizeof(NET_IN_SET_GROUPINFO_FOR_CHANNEL));
    memset(stOutChannelDeploy, 0, sizeof(NET_OUT_SET_GROUPINFO_FOR_CHANNEL));
    stInChannelDeploy.dwSize = sizeof(NET_IN_SET_GROUPINFO_FOR_CHANNEL);
    stOutChannelDeploy.dwSize = sizeof(NET_OUT_SET_GROUPINFO_FOR_CHANNEL);
    CLIENT_SetGroupInfoForChannel(ILoginHandle, &stInChannelDeploy, &stOutChannelDeploy);
}
```


3.4.4.2 以库为对象布控

```
// 输入参数
NET_IN_FACE_RECOGNITION_PUT_DISPOSITION_INFO stInFaceRecognitionDeploy =
{ sizeof(NET_IN_FACE_RECOGNITION_PUT_DISPOSITION_INFO);
strncpy(stInFaceRecognitionDeploy.szGroupId, strGroupId, DH_COMMON_STRING_64-1); // 需要布控的人脸库
stInFaceRecognitionDeploy.nDispositionChnNum = 2; // 布控的视频通道个数
stInFaceRecognitionDeploy.stuDispositionChnInfo[0].nChannelID = 0; // 人脸库部署通道
stInFaceRecognitionDeploy.stuDispositionChnInfo[0].nSimilary = 90; // 相似度阈值
stInFaceRecognitionDeploy.stuDispositionChnInfo[1].nChannelID = 2;
stInFaceRecognitionDeploy.stuDispositionChnInfo[1].nSimilary = 85;

// 输出参数
NET_OUT_FACE_RECOGNITION_PUT_DISPOSITION_INFO stOutFaceRecognitionDeploy =
{ sizeof(NET_OUT_FACE_RECOGNITION_PUT_DISPOSITION_INFO));

// 以人脸库为对象布控
bool nRet = CLIENT_FaceRecognitionPutDisposition(ILLoginHandle, &stInFaceRecognitionDeploy,
&stOutFaceRecognitionDeploy);
if(false == nRet)
{
    printf("CLIENT_FaceRecognitionPutDisposition: failed! Error code: %x.\n", CLIENT_GetLastError());
}

// 撤控输入参数，可对以部署的部分通道进行人脸库的撤控
NET_IN_FACE_RECOGNITION_DEL_DISPOSITION_INFO stInFaceRecognitionDel =
{ sizeof(NET_IN_FACE_RECOGNITION_DEL_DISPOSITION_INFO));
stInFaceRecognitionDel
strncpy(stInFaceRecognitionDel.szGroupId, strGroupId, DH_COMMON_STRING_64-1);
stInFaceRecognitionDel.nDispositionChnNum = 2; // 撤控的通道个数
stInFaceRecognitionDel.nDispositionChn[0] = 0; // 通道 1 撤控
stInFaceRecognitionDel.nDispositionChn[1] = 1;

// 撤控输出参数
NET_OUT_FACE_RECOGNITION_DEL_DISPOSITION_INFO stOutFaceRecognitionDel =
{ sizeof(NET_OUT_FACE_RECOGNITION_DEL_DISPOSITION_INFO));

bool nRet = CLIENT_FaceRecognitionDelDisposition(ILLoginHandle, &stInFaceRecognitionDel,
&stOutFaceRecognitionDel);
if(false == nRet)
```

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

3.5 以图搜图

3.5.1 简介

外部导入一张图片和相似度值，IVSS、NVR 等设备通过这张图检索历史库或人脸库是否已经存在匹配的人脸，并返回在相似度之上的图片结果。

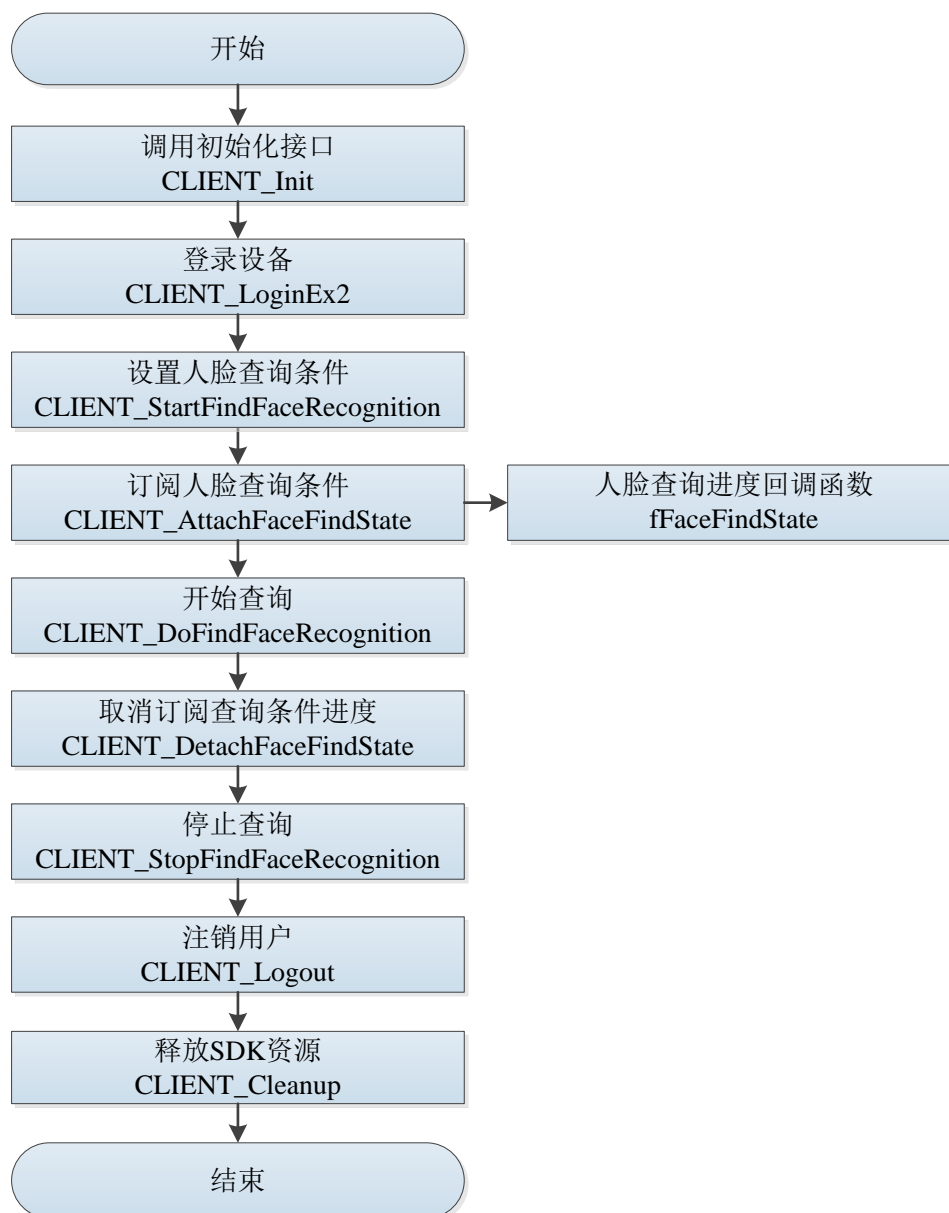
3.5.2 接口总览

接口	说明
CLIENT_StartFindFaceRecognition	设置人脸的查询条件
CLIENT_AttachFaceFindState	订阅人脸查询条件
CLIENT_DetachFaceFindState	取消订阅查询条件进度
CLIENT_DoFindFaceRecognition	开始查询
CLIENT_StopFindFaceRecognition	结束查询

3.5.3 流程说明

以图搜图流程，如图 3-4 所示。

图3-4 以图搜图流程



流程说明

- 步骤1 调用 `CLIENT_Init` 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 `CLIENT_LoginEx2` 登录设备。
- 步骤3 调用 `CLIENT_StartFindFaceRecognition` 设置人脸查询条件。
- 步骤4 查看步骤 3 中的返回值，若接口出参结构体中 `nTotalCount` 返回的是 -1，需要等待设备查询完成。
- 步骤5 调用 `CLIENT_AttachFaceFindState` 订阅人脸查询状态。然后等待进度回调函数中的返回进度为 100 时即搜索完成。搜索完成后调用 `CLIENT_DetachFaceFindState` 取消订阅查询进度。
- 步骤6 调用 `CLIENT_DoFindFaceRecognition` 接口获取查询结果。
- 步骤7 调用 `CLIENT_StopFindFaceRecognition` 接口结束查询。
- 步骤8 业务使用完后，调用 `CLIENT_Logout` 登出设备。
- 步骤9 SDK 功能使用完后，调用 `CLIENT_Cleanup` 释放 SDK 资源。

3.5.4 示例代码

```
// 查询进度回调函数
void CALLBACK FaceFindState(LLONG lLoginID, LLONG lAttachHandle, NET_CB_FACE_FIND_STATE*
pstStates, int nStateNum, LDWORD dwUser)
{
    if (pstStates->nProgress== 100) //表示查询进度 100%
    {
        // 停止人脸查询进度的订阅
        CLIENT_DetachFaceFindState(lAttachHandle);
        // 开始查询
        DoFind();
    }
    return;
}

// 配置查询条件
NET_IN_STARTFIND_FACERECONGNITION stuInParam = { sizeof(stuInParam) };
NET_OUT_STARTFIND_FACERECONGNITION stuOutParam = { sizeof(stuOutParam) };
stuInParam.stFilterInfo.dwSize = sizeof(stuInParam.stFilterInfo);
stuInParam.stMatchOptions.dwSize = sizeof(stuInParam.stMatchOptions);
stuInParam.bPersonExEnable = TRUE;
stuInParam.nChannelID = 0;
stuInParam.stMatchOptions.nSimilarity = 80;
stuInParam.stFilterInfo.stStartTime = startTime;
stuInParam.stFilterInfo.stEndTime = endTime;
stuInParam.nBufferLen = nPicBufLen;
stuInParam.pBuffer = strPicBuf; // 图片 Buffer
stuInParam.stPersonInfoEx.wFacePicNum = 1;
stuInParam.stPersonInfoEx.szFacePicInfo[0].dwOffSet = 0;
stuInParam.stPersonInfoEx.szFacePicInfo[0].dwFileLenth = nLength;

BOOL bRet = CLIENT_StartFindFaceRecognition(m_lLoginId, &stuInParam, &stuOutParam, 5000);
if (!bRet)
{
    printf("CLIENT_StartFindFaceRecognition: failed! Error code %x.\n", CLIENT_GetLastError());
    return -1;
}
m_lFindHandle = stuOutParam.lFindHandle;
```

```

if (-1 == stuOutParam.nTotalCount)
{
    // 当查询总数为-1 时，表示设备未查询完毕，需要订阅设备查询进度
    NET_IN_FACE_FIND_STATE stuInFindState = { sizeof(stuInFindState) };
    NET_OUT_FACE_FIND_STATE stuOutFindState = { sizeof(stuOutFindState) };
    stuInFindState.nTokenNum = 1;
    int nToken = stuOutParam.nToken;
    stuInFindState.nTokens = &nToken;
    stuInFindState.cbFaceFindState = FaceFindState; // 进度回调函数
    stuInFindState.dwUser = (DWORD)this;
    m_lAttachHandle = CLIENT_AttachFaceFindState(m_lLoginId, &stuInFindState, &stuOutFindState, 5000);
}
else
{
    // 直接开始查询
    DoFind();
}

void DoFind()
{
    NET_IN_DOFIND_FACERECONGNITION          stuInDoFind          =
{ sizeof(NET_IN_DOFIND_FACERECONGNITION) };
    NET_OUT_DOFIND_FACERECONGNITION          stuOutDoFind         =
{ sizeof(NET_OUT_DOFIND_FACERECONGNITION) };
    stuOutDoFind.bUseCandidatesEx = TRUE;
    stuInDoFind.nCount = 20; // 每次查询 20 条信息,本次查询的总数大于 20
    stuInDoFind.lFindHandle = m_lFindHandle;
    stuInDoFind.emDataType = EM_NEEDED_PIC_TYPE_HTTP_URL; // 指定查询结果返回图片的格式为
http 链接
    stuInDoFind.nBeginNum = 0; // 从 0 开始查询

    BOOL bRet = CLIENT_DoFindFaceRecognition(&stuInDoFind, &stuOutDoFind, 10000);
    if (!bRet)
    {
        printf("CLIENT_DoFindFaceRecognition: failed! Error code %x.\n", CLIENT_GetLastError());
        return ;
    }

    CLIENT_StopFindFaceRecognition(m_lFindHandle);
}

```

3.6 人脸相关录像图片查询与下载

因为人脸相关的智能为智能事件的一种，因此人脸相关的录像图片查询请参见“2.5 查询/回放/下载录像和图片”。

在查询时，调用接口 `CLIENT_FindFileEx` 设置查询条件，关于人脸相关的查询操作，接口参数 `emType` 的取值情况如下：

- `DH_FILE_QUERY_FACE` 查询人脸识别图片
- `DH_FILE_QUERY_FACE_DETECTION` 查询人脸检测图片
- `DH_FILE_QUERY_FILE` 查询录像，且接口参数 `pQueryCondition` 为类型

`NET_IN_MEDIA_QUERY_FILE` 的结构体指针，结构体中的 `nEventLists` 字段为：

- `EVENT_IVS_FACERECOGNITION` 人脸识别录像查询
- `EVENT_IVS_FACEDETECT` 人脸检测录像查询

4.1 人体事件订阅

具体的订阅请参见“2.4 智能事件订阅”，在智能事件上报的回调函数 `fAnalyzerDataCallBack` 中过滤出人体检测事件，即 `EVENT_IVS_HUMANTRAIT` 人体检测事件。

4.2 人体图片搜索

4.2.1 简介

人体检测的图片搜索的代码参考“2.5 查询/回放/下载录像和图片”中的图片搜索，以下为人体检测的图片下载的介绍。

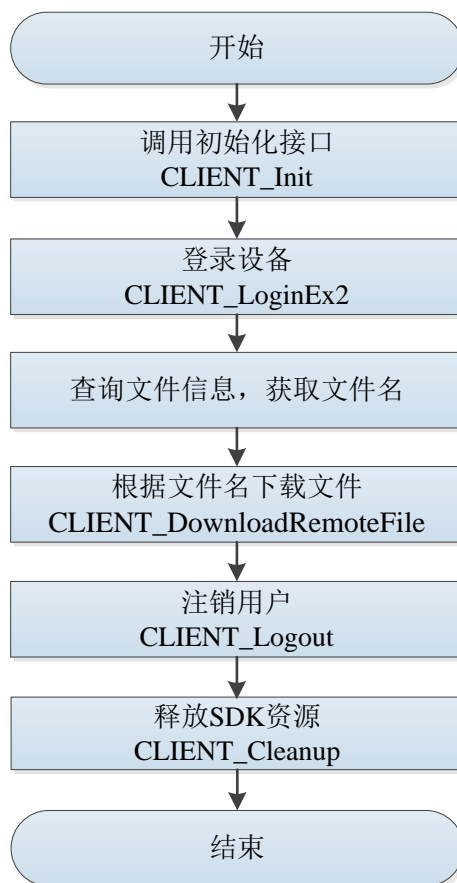
4.2.2 接口总览

接口	说明
<code>CLIENT_DownloadRemoteFile</code>	下载文件

4.2.3 流程说明

人体图片搜索流程，如图 4-1 所示。

图4-1 人体图片搜索流程



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用 SDK 接口查询文件信息，获取需要下载文件的文件名。
- 步骤4 使用查询到的文件信息，调用接口 CLIENT_DownloadRemoteFile 下载文件。
- 步骤5 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤6 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

4.2.4 示例代码

```
DH_IN_DOWNLOAD_REMOTE_FILE stuInDownloadFile =
{sizeof(DH_IN_DOWNLOAD_REMOTE_FILE )};
stuInDownloadFile.pszFileName = strFileName
stuInDownloadFile.pszFileDst = strDownloadName;

DH_OUT_DOWNLOAD_REMOTE_FILE stuOutDownloadFile =
{sizeof(DH_OUT_DOWNLOAD_REMOTE_FILE )};
BOOL bRet = CLIENT_DownloadRemoteFile(m_lLoginHandle, &stuInDownloadFile, &stuOutDownloadFile);
if (bRet == FALSE)
{
```



```
MessageBox(ConvertString("Download Failed"), ConvertString("Prompt"));
return;
}
```

5.1 客流量事件订阅

5.1.1 简介

客流量统计功能指在经营区域安装前端设备，智能分析服务器根据前端采集的视频数据精确统计出每个入口实时客流进出人数。该类产品被广泛应用于大型商业、旅游行业、公共安全、文博和连锁等行业。

客流量数据的实时订阅，可以获取到当天总出入人数情况和实时出入的人数的统计情况。

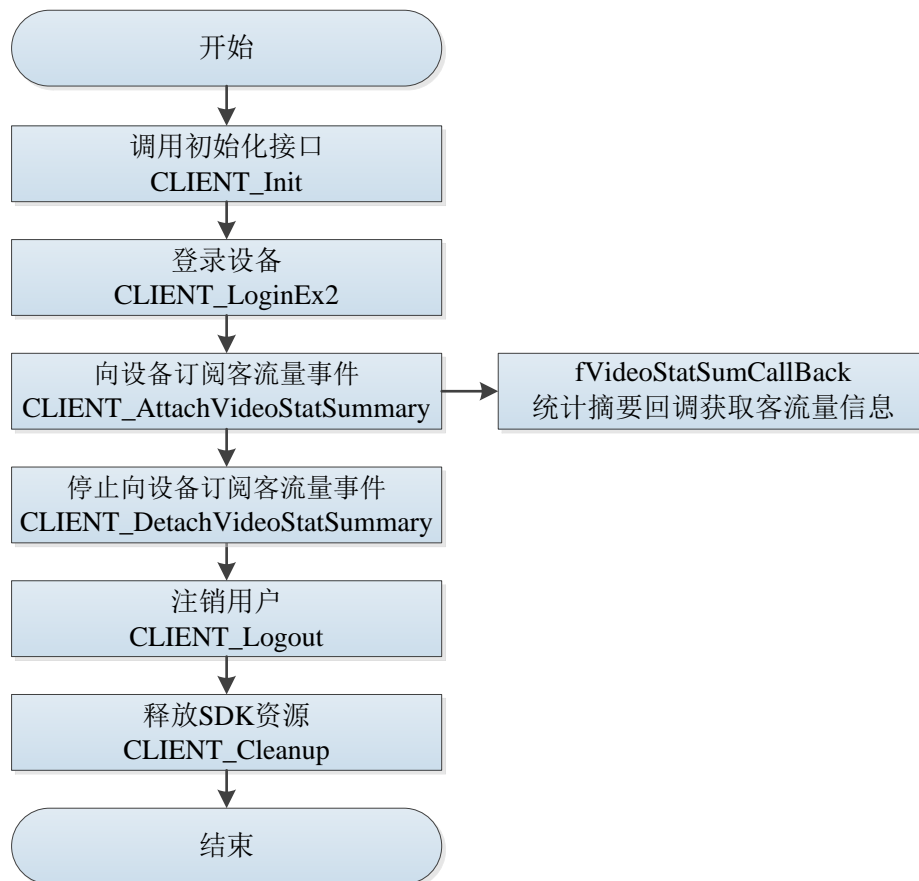
5.1.2 接口总览

接口	说明
CLIENT_AttachVideoStatSummary	订阅客流量事件
CLIENT_DetachVideoStatSummary	退订客流量事件

5.1.3 流程说明

客流量订阅流程，如图 5-1 所示。

图5-1 客流量订阅流程



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用 CLIENT_AttachVideoStatSummary 向设备订阅客流量事件。
- 步骤4 订阅成功后，设备上报的客流量事件通过 fVideoStatSumCallBack 回调函数获取人脸事件并通知给用户。
- 步骤5 客流量事件上报功能使用完毕后，调用 CLIENT_DetachVideoStatSummary 停止订阅客流量事件。
- 步骤6 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤7 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

5.1.4 示例代码

```

void CALLBACK VideoStatSumCallback(LLONG lAttachHandle, NET_VIDEOSTAT_SUMMARY* pBuf,
DWORD dwBufLen, LDWORD dwUser)
{
    // 处理回调数据
}

NET_IN_ATTACH_VIDEOSTAT_SUM InParam = {sizeof(NET_IN_ATTACH_VIDEOSTAT_SUM)};
NET_OUT_ATTACH_VIDEOSTAT_SUM OutParam = {sizeof(NET_OUT_ATTACH_VIDEOSTAT_SUM)};
    
```

```
InParam.nChannel=0;
InParam.cbVideoStatSum=VideoStatSumCallback; // 订阅回调函数

// 订阅客流量统计
LLONG attachHnd = CLIENT_AttachVideoStatSummary(ILLoginID,&InParam,&OutParam,5000)
if(0 == attachHnd)
{
    printf("CLIENT_AttachVideoStatSummary failed! Error code %x.\n", CLIENT_GetLastError());
    return;
}

// 取消订阅客流量统计
CLIENT_DetachVideoStatSummary(attachHnd);
```

5.2 客流量事件报警

具体的订阅请参见“2.4 智能事件订阅”，在智能事件上报的回调函数 fAnalyzerDataCallBack 中过滤出客流量智能事件，即“EVENT_IVS_NUMBERSTAT 人数统计事件”和“EVENT_IVS_MAN_NUM_DETECTION 区域人数统计事件”。

5.3 客流量历史数据查询

5.3.1 简介

用户指定客流量信息的起始时间和结束时间，设备端将查询结果返回给 SDK。

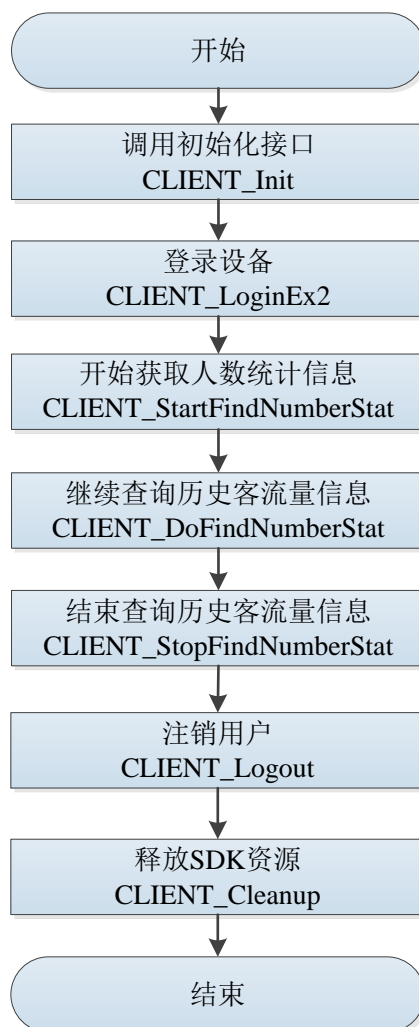
5.3.2 接口总览

接口	说明
CLIENT_StartFindNumberStat	开始查询历史客流量信息
CLIENT_DoFindNumberStat	继续查询历史客流量信息
CLIENT_StopFindNumberStat	结束查询历史客流量信息

5.3.3 流程说明

客流图片录像的查询流程，如图 5-2 所示。

图5-2 录像或图片的查询流程



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用 CLIENT_StartFindNumberStat 开始获取人数统计信息。
- 步骤4 调用 CLIENT_DoFindNumberStat 继续查询某段时间的客流量统计信息。
- 步骤5 调用 CLIENT_StopFindNumberStat 停止记录查询。
- 步骤6 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤7 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

5.3.4 示例代码

```
// 设置查询条件
NET_IN_FINDNUMBERSTAT inParam = { sizeof(NET_IN_FINDNUMBERSTAT)};
inParam.nChannelID = nChannelID; // 要进行查询的通道号
inParam.nGranularityType = 1; // 查询粒度 0: 分钟, 1: 小时, 2: 日, 3: 周, 4: 月, 5: 季, 6: 年
inParam.nWaittime = 5000; // 等待接收数据的超时时间
NET_OUT_FINDNUMBERSTAT outParam { sizeof(NET_OUT_FINDNUMBERSTAT)};
LLONG findHnd = CLIENT_StartFindNumberStat(pLoginHandle, &inParam, &outParam);
```

```

if (findHand == 0)
{
    printf("CLIENT_StartFindNumberStat failed! Error code %x.\n", CLIENT_GetLastError());
    return ;
}

NET_IN_DOFINDNUMBERSTAT inDoFind = {sizeof(NET_IN_DOFINDNUMBERSTAT)};
NET_OUT_DOFINDNUMBERSTAT outDoFind = {sizeof(NET_OUT_DOFINDNUMBERSTAT)};
inDoFind.nBeginNumber = 0; // 从 0 开始查询
inDoFind.nCount = 10; // 每次查询 10 条
inDoFind.nWaittime = 5000; // 接口超时时间 5s

outDoFind.pstuNumberStat = new DH_NUMBERSTAT[10];
outDoFind.nBufferLen = 10 * sizeof(DH_NUMBERSTAT);
for (int i = 0; i < 10 ; i++)
{
    outDoFind.pstuNumberStat[i].dwSize = sizeof(DH_NUMBERSTAT);
}

// 查询
BOOL bRet = CLIENT_DoFindNumberStat(findHand, &inDoFind, &outDoFind)
if (FALSE == bRet)
{
    printf("CLIENT_DoFindNumberStat failed! Error code %x.\n", CLIENT_GetLastError());
    delete[] outDoFind.pstuNumberStat;
    return ;
}

// 停止查询客流量
CLINET_StopFindNumberStat(findHand);
delete[] outDoFind.pstuNumberStat;

```

6.1 通用行为事件的订阅

具体的订阅请参见“2.4 智能事件订阅”，在智能事件上报的回调函数 `fAnalyzerDataCallBack` 中过滤出通用行为事件，如下：

- `EVENT_IVS_CROSSLINEDETECTION` 绊线入侵事件
- `EVENT_IVS_CROSSREGIONDETECTION` 区域入侵事件

6.2 通用行为事件录像查询和下载

因为通用行为为智能事件的一种，因此相关的录像图片查询请参见“2.5 查询/回放/下载录像和图片”。

在查询时，调用接口 `CLIENT_FindFileEx` 设置查询条件，关于通用行为录像的查询操作，接口参数 `emType` 的取值情况如下：

- `DH_FILE_QUERY_FILE` 查询录像，且接口参数 `pQueryCondition` 为类型 `NET_IN_MEDIA_QUERY_FILE` 的结构体指针，结构体中的 `nEventLists` 字段为：
 - `EVENT_IVS_CROSSLINEDETECTION` 绊线入侵录像查询
 - `EVENT_IVS_CROSSREGIONDETECTION` 区域入侵录像查询

7.1 智能交通事件订阅

具体的订阅请参见“2.4 智能事件订阅”，在智能事件上报的回调函数 `fAnalyzerDataCallBack` 中过滤出智能交通事件，如下：

- `EVENT_IVS_TRAFFICJUNCTION` 交通路口事件
- `EVENT_IVS_TRAFFICJAM` 交通拥堵事件
- `EVENT_IVS_TRAFFIC_OVERSPEED` 超速事件
- `EVENT_IVS_TRAFFIC_UNDERSPEED` 欠速
- `EVENT_IVS_TRAFFIC_PEDESTRAIN` 行人事件
- `EVENT_IVS_TRAFFIC_FLOWSTATE` 车流量事件

7.2 车流量历史数据查询

7.2.1 简介

查询车流量的历史数据。

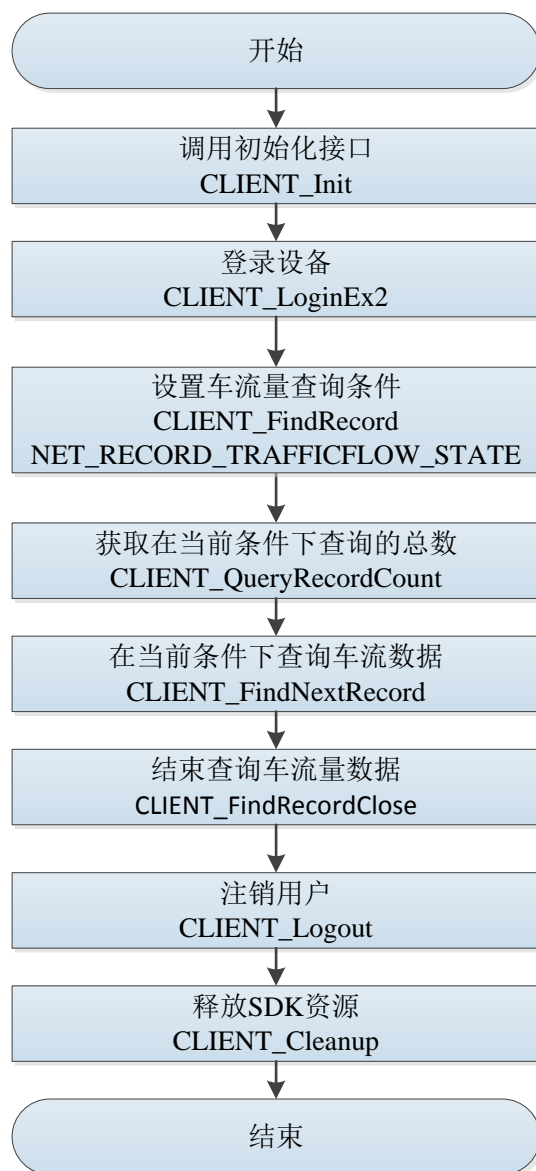
7.2.2 接口总览

接口	说明
<code>CLIENT_FindRecord</code>	设置查询条件
<code>CLIENT_QueryRecordCount</code>	获取查询数量
<code>CLIENT_FindNextRecord</code>	在当前查询条件下查询数据
<code>CLIENT_FindRecordClose</code>	关闭查询

7.2.3 流程说明

车流量历史数据查询流程，如图 7-1 所示。

图7-1 车流量历史数据查询



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用接口 CLIENT_FindRecord 设置车流量的查询条件，使用的枚举为 NET_RECORD_TRAFFICFLOW_STATE。
- 步骤4 调用接口 CLIENT_QueryRecordCount，获取在当前查询条件下可以查到的数据总数。
- 步骤5 调用接口 CLIENT_FindNextRecord，在当前查询条件下查询指定条数的车流数据。
- 步骤6 查询结束后，调用接口 CLIENT_FindRecordClose 清理查询资源。
- 步骤7 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤8 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- 在查询车流量过程中，首先，设备必须支持该功能，且在查询时间段内有车流数据。此外，需要设备有 SD 卡，这样可以保存车流数据。

- 在查询的车流中，平均速度字段如果为-1，则表示该时间段没有车辆经过；如果大于 0，则表示车辆的平均速度；如果为 0，则表示平均速度为 0。

7.2.4 示例代码

```
// 开始查询，设置查询条件
FIND_RECORD_TRAFFICFLOW_CONDITION stTrafficFlow =
{sizeof(FIND_RECORD_TRAFFICFLOW_CONDITION)};
stTrafficFlow.abChannelId = TRUE;
stTrafficFlow.nChannelId = 0;
stTrafficFlow.abLane = FALSE;
stTrafficFlow.bStartTime = TRUE;
stTrafficFlow.bEndTime = TRUE;
stTrafficFlow.stStartTime = startTime;
stTrafficFlow.stEndTime = endTime;
stTrafficFlow.bStatisticsTime = TRUE;

NET_IN_FIND_RECORD_PARAM stuFindInParam = {sizeof(NET_IN_FIND_RECORD_PARAM)};
stuFindInParam.emType = NET_RECORD_TRAFFICFLOW_STATE;
stuFindInParam.pQueryCondition = &stTrafficFlow;

NET_OUT_FIND_RECORD_PARAM stuFindOutParam = {sizeof(NET_OUT_FIND_RECORD_PARAM)};
bool bRet = CLIENT_FindRecord(m_lLoginHandle, &stuFindInParam, &stuFindOutParam, MAX_TIMEOUT);
if (!bRet)
{
    return;
}

// 查询总数
NET_IN_QUEYT_RECORD_COUNT_PARAM inQueryCountParam =
{ sizeof(NET_IN_QUEYT_RECORD_COUNT_PARAM)};
inQueryCountParam.lFindeHandle = stuFindOutParam.lFindeHandle;
NET_OUT_QUEYT_RECORD_COUNT_PARAM outQueryCountParam =
{ sizeof(NET_OUT_QUEYT_RECORD_COUNT_PARAM) };

bRet = CLIENT_QueryRecordCount(&inQueryCountParam, &outQueryCountParam, MAX_TIMEOUT);
if (!bRet)
{
    MessageBox(ConvertString("Query record count failed!"), ConvertString("Prompt"));
    return;
}
```

```

// 查询 100 条数据
int nQueryCount = 100;
NET_RECORD_TRAFFIC_FLOW_STATE* pRecordList = new
NET_RECORD_TRAFFIC_FLOW_STATE[nQueryCount];
memset(pRecordList, 0, sizeof(NET_RECORD_TRAFFIC_FLOW_STATE) * nQueryCount);
for (int unIndex = 0; unIndex < nQueryCount; ++unIndex)
{
    pRecordList[unIndex].dwSize = sizeof(NET_RECORD_TRAFFIC_FLOW_STATE);
}

NET_IN_FIND_NEXT_RECORD_PARAM stuFindNextInParam =
{sizeof(NET_IN_FIND_NEXT_RECORD_PARAM)};
stuFindNextInParam.lFindeHandle = stuFindOutParam.lFindeHandle;
stuFindNextInParam.nFileCount = nQueryCount;

NET_OUT_FIND_NEXT_RECORD_PARAM stuFindNextOutParam =
{sizeof(NET_OUT_FIND_NEXT_RECORD_PARAM)};
stuFindNextOutParam.pRecordList = pRecordList;
stuFindNextOutParam.nMaxRecordNum = nQueryCount;
bRet = CLIENT_FindNextRecord(&stuFindNextInParam, &stuFindNextOutParam, MAX_TIMEOUT);
if (!bRet)
{
    MessageBox(ConvertString("Query record count failed!"), ConvertString("Prompt"));
}

// 结束查询
CLIENT_FindRecordClose(stuFindOutParam.lFindeHandle);
delete[] pRecordList;

```

7.3 车辆黑白名单增删改查

7.3.1 简介

对车辆的黑白名单进行增删改查操作。

7.3.2 接口总览

接口	说明
CLIENT_OperateTrafficList	黑白名单的增删改操作
CLIENT_FindRecord	设置查询条件
CLIENT_QueryRecordCount	获取查询数量
CLIENT_FindNextRecord	在当前查询条件下查询数据
CLIENT_FindRecordClose	关闭查询

7.3.3 流程说明

车辆黑白名单增删改查流程，如图 7-2 所示。

图7-2 车辆黑白名单增删改查



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用接口 CLIENT_OperateTrafficList，枚举类型为 NET_TRAFFIC_LIST_INSERT，增加

黑白名单。

- 步骤4 调用接口 `CLIENT_FindRecord` 设置黑白名单的查询条件，使用的枚举为 `NET_RECORD_TRAFFICREDLIST`(白名单)、`NET_RECORD_TRAFFICBLACKLIST` (黑名单)。
- 步骤5 调用接口 `CLIENT_QueryRecordCount`，获取在当前查询条件下可以查到的数据总数。
- 步骤6 调用接口 `CLIENT_FindNextRecord`，在当前查询条件下查询指定条数的黑白名单数量。
- 步骤7 使用查询到的黑白名单信息，调用接口 `CLIENT_OperateTrafficList` 进行黑白名单的修改删除操作，枚举为 `NET_TRAFFIC_LIST_REMOVE` (删除)、`NET_TRAFFIC_LIST_UPDATE` (修改)。
- 步骤8 查询结束后，调用接口 `CLIENT_FindRecordClose` 清理查询资源。
- 步骤9 业务使用完后，调用 `CLIENT_Logout` 登出设备。
- 步骤10 SDK 功能使用完后，调用 `CLIENT_Cleanup` 释放 SDK 资源。

注意事项

调用接口 `CLIENT_OperateTrafficList` 增加黑白名单的操作时，当接口返回-1，不表示接口失败，而是表示还未生成记录集号。因为设备将增加的名单先保存到缓冲中，就返回添加成功的结果，之后再缓冲的数据保存到数据库中。在设备返回结果的时候，因为名单还未添加到数据库中，因此没有记录集编号，设备那边就返回-1 表示。

7.3.4 示例代码

查询黑白名单的代码请参见“7.2.4 示例代码”，以下为黑白名单的增删改查的示例代码。

```
// 黑白名单的增加
NET_IN_OPERATE_TRAFFIC_LIST_RECORD stInParam =
{ sizeof(NET_IN_OPERATE_TRAFFIC_LIST_RECORD) };
NET_OUT_OPERATE_TRAFFIC_LIST_RECORD stOutParam =
{ sizeof(NET_OUT_OPERATE_TRAFFIC_LIST_RECORD) };
stInParam.emOperateType = NET_TRAFFIC_LIST_INSERT;
stInParam.emRecordType = NET_RECORD_TRAFFICBLACKLIST; // 黑名单
//stInParam.emRecordType = NET_RECORD_TRAFFICREDLIST; // 白名单

NET_TRAFFIC_LIST_RECORD stTrafficListRecord = { sizeof(NET_TRAFFIC_LIST_RECORD) };
stTrafficListRecord.stBeginTime = startTime;
stTrafficListRecord.stCancelTime = endTime;
strncpy(stTrafficListRecord.szPlateNumber, strPlateNumber.GetBuffer(),
DH_MAX_PLATE_NUMBER_LEN-1);
strncpy(stTrafficListRecord.szMasterOfCar, strOwner.GetBuffer(), DH_MAX_NAME_LEN-1);

NET_INSERT_RECORD_INFO stInsertInfo = { sizeof( NET_INSERT_RECORD_INFO ) };
stInsertInfo.pRecordInfo = &stTrafficListRecord;
stInParam.pstOpreateInfo = &stInsertInfo;
```

```

bool bRet = CLIENT_OperateTrafficList(m_lLoginHandle, &stInParam, &stOutParam, MAX_TIMEOUT);
if (!bRet)
{
    return;
}

// 黑白名单的修改
NET_IN_OPERATE_TRAFFIC_LIST_RECORD stInParam =
{ sizeof(NET_IN_OPERATE_TRAFFIC_LIST_RECORD) };
NET_OUT_OPERATE_TRAFFIC_LIST_RECORD stOutParam =
{ sizeof(NET_OUT_OPERATE_TRAFFIC_LIST_RECORD) };

stInParam.emOperateType = NET_TRAFFIC_LIST_UPDATE;
stInParam.emRecordType = NET_RECORD_TRAFFICBLACKLIST; // 黑名单
//stInParam.emRecordType = NET_RECORD_TRAFFICREDLIST; // 白名单

NET_TRAFFIC_LIST_RECORD stTrafficListRecord = { sizeof(NET_TRAFFIC_LIST_RECORD) };
stTrafficListRecord.stBeginTime = startTime;
stTrafficListRecord.stCancelTime = endTime;
strncpy(stTrafficListRecord.szPlateNumber, strPlateNumber.GetBuffer(),
DH_MAX_PLATE_NUMBER_LEN-1);
strncpy(stTrafficListRecord.szMasterOfCar, strOwner.GetBuffer(), DH_MAX_NAME_LEN-1);
stTrafficListRecord.nRecordNo = m_stTrafficListInfo.nRecordNo; // 记录集编号

NET_UPDATE_RECORD_INFO stModifyRecord = { sizeof(NET_UPDATE_RECORD_INFO) };
stModifyRecord.pRecordInfo = &stTrafficListRecord;
stInParam.pstOpreateInfo = &stModifyRecord;

bool bRet = CLIENT_OperateTrafficList(m_lLoginHandle, &stInParam, &stOutParam, MAX_TIMEOUT);
if (!bRet)
{
    return;
}

// 黑白名单的删除
NET_REMOVE_RECORD_INFO stRemoveRecord = { sizeof(NET_REMOVE_RECORD_INFO) };
stRemoveRecord.nRecordNo = m_vecTrafficListInfo[nSelect]->nRecordNo; // 记录集编号

NET_IN_OPERATE_TRAFFIC_LIST_RECORD stInParam =

```

```
{ sizeof(NET_IN_OPERATE_TRAFFIC_LIST_RECORD) };
stInParam.emOperateType = NET_TRAFFIC_LIST_REMOVE;
stInParam.emRecordType = NET_RECORD_TRAFFICBLACKLIST; // 黑名单
//stInParam.emRecordType = NET_RECORD_TRAFFICREDLIST; // 白名单

stInParam.pstOpreateInfo = &stRemoveRecord;
NET_OUT_OPERATE_TRAFFIC_LIST_RECORD stOutParam =
{ sizeof(NET_OUT_OPERATE_TRAFFIC_LIST_RECORD) };

bool bRet = CLIENT_OperateTrafficList(m_lLoginHandle, &stInParam, &stOutParam, MAX_TIMEOUT);
if (!bRet)
{
    return;
}
```

7.4 车辆相关图片的查询与下载

7.4.1 简介

智能交通中会将智能事件的抓拍图片保存到设备的存储器上，可使用车牌或者相应智能事件等来搜索对应的图片，并支持对图片的下载。

车辆相关图片搜索请参见“2.5 查询/回放/下载录像和图片”中的图片查询内容，且接口 CLIENT_FindFileEx 查询的类型为 DH_FILE_QUERY_TRAFFICCAR_EX。

以下为查询到图片下载使用的接口及示例代码。

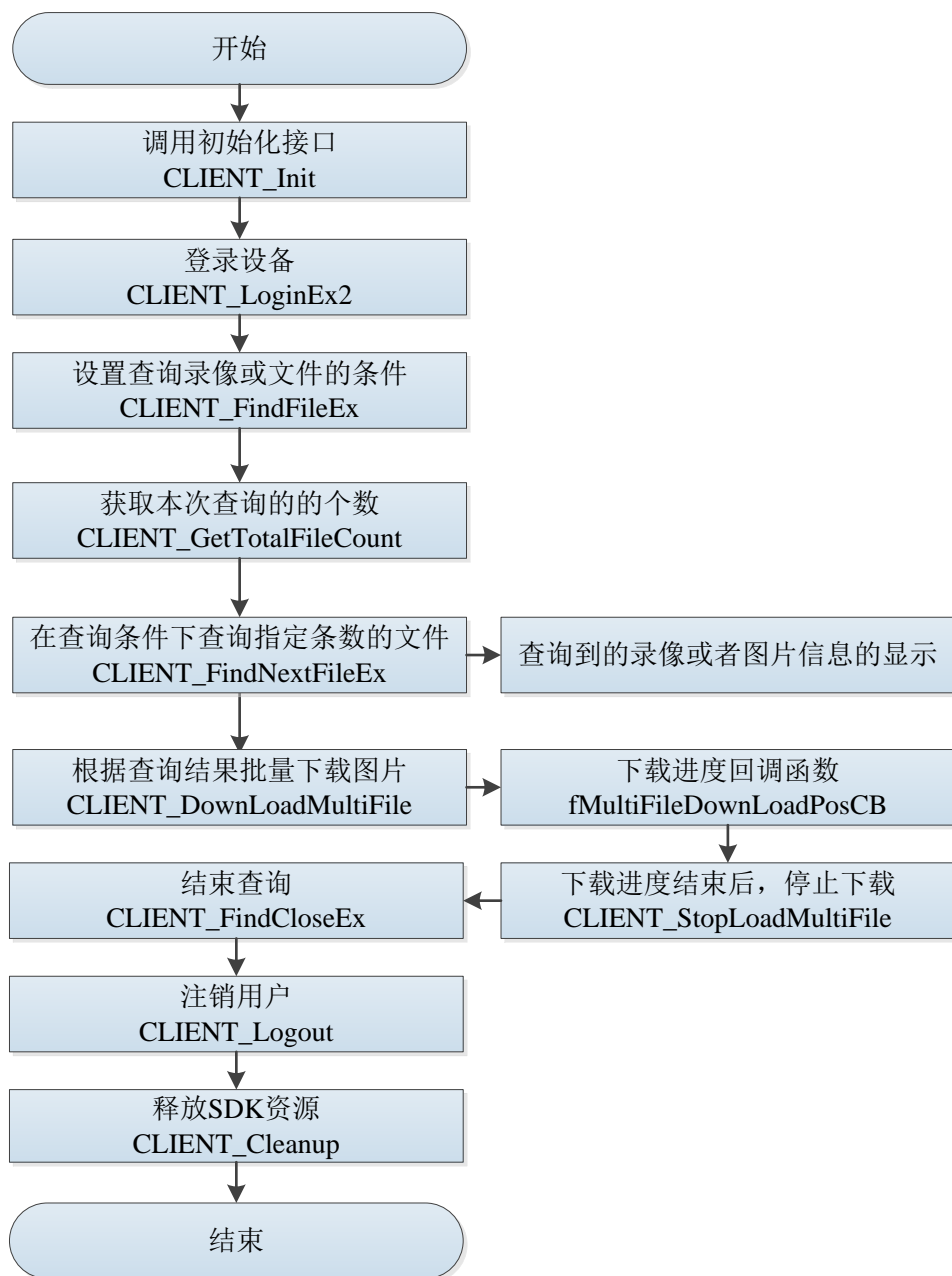
7.4.2 接口总览

接口	说明
CLIENT_DownLoadMultiFile	批量下载文件
CLIENT_StopLoadMultiFile	停止批量下载文件

7.4.3 流程说明

图片查询与下载流程，如图 7-3 所示。

图7-3 图片查询与下载过程



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用 CLIENT_FindFileEx 设置查询条件，设置成功后返回查询句柄。根据 emType 的不同取值，判断查找的类型。
- 步骤4 调用 CLIENT_GetTotalFileCount 获取查询到的录像或者文件总数。
- 步骤5 调用 CLIENT_FindNextFileEx 查询指定条数的录像或者图片，并将查询到的信息保存，用于录像或者图片的回放下载操作。
- 步骤6 使用查询到的文件信息，调用接口 CLIENT_DownloadMultiFile 批量下载文件。
- 步骤7 下载进度回调函数 fMultiFileDownloadPosCB 中的参数 dwDownloadSize 为最大值时，调用接口 CLIENT_StopLoadMultiFile 结束图片下载。
- 步骤8 调用 CLIENT_FindCloseEx 结束查询。
- 步骤9 业务使用完后，调用 CLIENT_Logout 登出设备。

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

注意事项

- 接口 `CLIENT_FindFileEx` 中参数 `pQueryCondition` 是由用户申请和释放的，具体类型由 `emType` 的枚举类型来定义。
- `CLIENT_FindFileEx` 如果查询成功，则会返回查询句柄，`CLIENT_FindNextFileEx` 将查询句柄作为参数查询具体的录像或者图片，且必须调用 `CLIENT_FindCloseEx` 将查询句柄关闭。
- `CLIENT_FindNextFileEx` 可设置查询条数，如果设置条数大于 1 条则参数 `pMediaFileInfo` 必须为一个数据指针，数据大小大于等于设置的条数。

7.4.4 示例代码

```
// 批量下载进度回调函数
void CALLBACK DownloadTrafficPicture(LLONG lDownloadHandle, DWORD dwID, DWORD
dwFileTotalSize, DWORD dwDownloadSize, int nError, LDWORD dwUser, void* pReserved)
{
    if (nError != 0 || dwDownloadSize == UINT_MAX)
    {
        // 下载出错或者下载结束
        CLIENT_StopLoadMultiFile(m_lDownloadHandle);
        delete[] pDownloadInfo;
    }
}

NET_DOWNLOADFILE_INFO* pDownloadInfo = new NET_DOWNLOADFILE_INFO[10]; //下载 10 文件
memset(pDownloadInfo, 0, 10*sizeof(NET_DOWNLOADFILE_INFO));
for(int i=0; i++; i<10)
{
    pDownloadInfo[i].dwFileID = 1;

    // stTrafficPicture 的数据是通过 CLIENT_FindFileEx 查询出来的文件信息中的文件，类型为
    MEDIAFILE_TRAFFICCAR_INFO_EX
    pDownloadInfo[i].nFileSize = stTrafficPicture.stuInfo.sizeEx / 1024;
    strncpy(pDownloadInfo[i].szSourceFilePath, stTrafficPicture.stuInfo.szFilePath, MAX_PATH-1);

    // 下载后的文件保存路径
    strncpy(pDownloadInfo[i].szSavedFileName, szFilePathName[i], MAX_PATH-1);
}

NET_IN_DOWNLOAD_MULTI_FILE stInDownloadFile = { sizeof(NET_IN_DOWNLOAD_MULTI_FILE )};
NET_OUT_DOWNLOAD_MULTI_FILE stOutDownloadFile=
```

```

{sizeof(NET_OUT_DOWNLOAD_MULTI_FILE )});
stInDownloadFile.emDownloadType = EM_DOWNLOAD_BY_FILENAME;
stInDownloadFile.cbPosCallBack = DownloadTrafficPicture; // 批量下载进度回调函数
stInDownloadFile.dwUserData = (LDWORD)this;
stInDownloadFile.nFileCount = 10; // 下载文件个数
stInDownloadFile.pFileInfos = pDownLoadInfo; // 需要下载的文件信息指针，如需下载多个则为数组首地址

// 批量下载图片文件
BOOL nRet = CLIENT_DownloadMultiFile(m_lLoginHandle, &stInDownloadFile, &stOutDownloadFile,
MAX_TIMEOUT);
if (!nRet)
{
    printf("CLIENT_DownloadMultiFile failed! Error code %x.\n", CLIENT_GetLastError());
    delete[] pDownLoadInfo;
    return;
}
m_lDownloadHandle = stOutDownloadFile.lDownloadHandle;

```

8.1 门禁事件订阅

具体的订阅请参见“2.4 智能事件订阅”，在智能事件上报的回调函数 `fAnalyzerDataCallBack` 中过滤出门禁事件，如下：

`EVENT_IVS_ACCESS_CTL` 门禁事件

8.2 门禁卡信息管理

8.2.1 简介

对门禁卡信息如卡号、用户 ID、卡名等相关信息进行增删改查的操作。

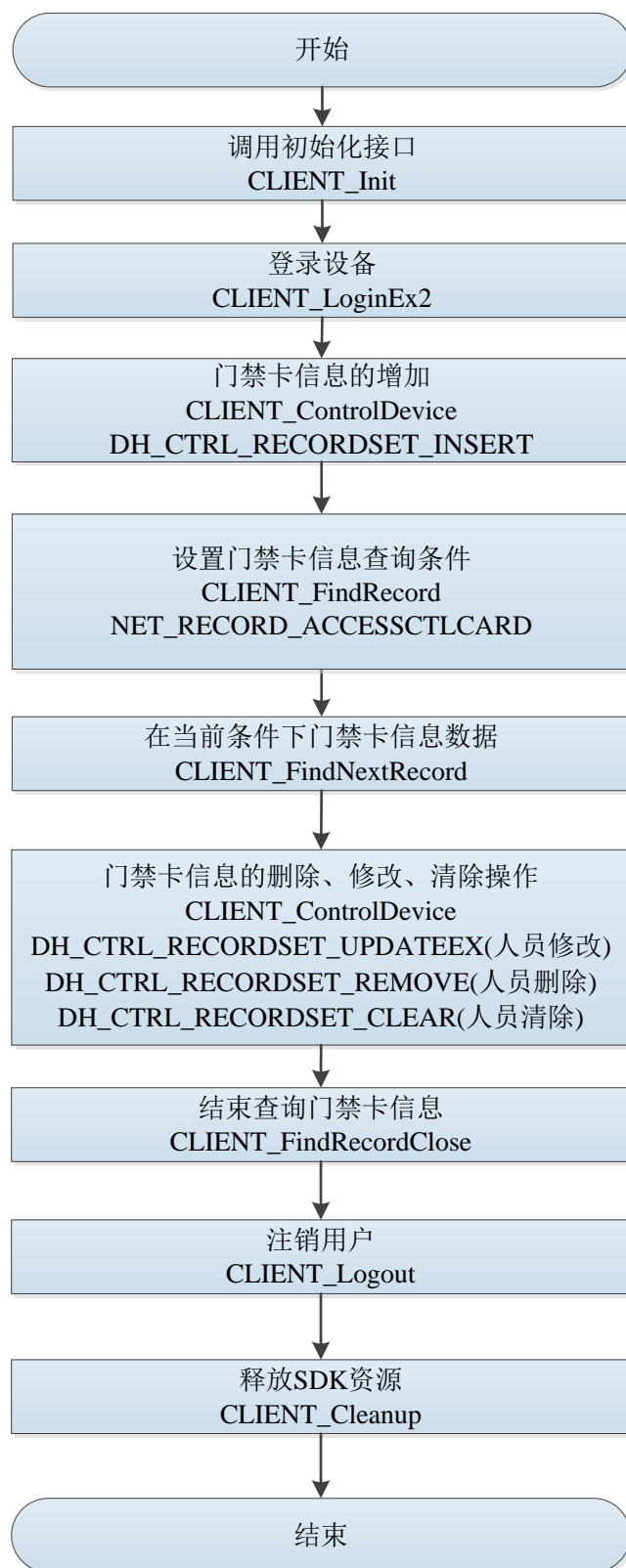
8.2.2 接口总览

接口	说明
<code>CLIENT_FindRecord</code>	设置查询条件
<code>CLIENT_FindNextRecord</code>	在当前查询条件下查询数据
<code>CLIENT_FindRecordClose</code>	关闭查询
<code>CLIENT_ControlDevice</code>	门禁卡信息的增加、修改、删除、清除

8.2.3 流程说明

门禁卡信息的增删改查流程，如图 8-1 所示。

图8-1 门禁卡信息的增删改查



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用 CLIENT_ControlDevice 增加门禁卡信息，使用的枚举为

- DH_CTRL_RECORDSET_INSERT。
- 步骤4 调用接口 CLIENT_FindRecord 设置门禁卡信息的查询条件，使用的枚举为 NET_RECORD_ACCESSCTLCARD。
- 步骤5 调用接口 CLIENT_FindNextRecord，在当前查询条件下查询指定条数的门禁卡信息数据。
- 步骤6 查询结束后，调用接口 CLIENT_FindRecordClose 清理查询资源。
- 步骤7 调用 CLIENT_ControlDevice，修改门禁卡信息，使用的枚举为 NET_RECORD_ACCESSCTLCARD。
- 步骤8 调用 CLIENT_ControlDevice，删除门禁卡信息，使用的枚举为 DH_CTRL_RECORDSET_REMOVE。
- 步骤9 调用 CLIENT_ControlDevice，清除门禁卡信息，使用的枚举为 DH_CTRL_RECORDSET_CLEAR。
- 步骤10 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤11 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- 新增门禁卡的时候，需要保证 CardNo 和 UserID 与之前保存在设备的门禁卡信息不重复，否则会添加失败。
- 删除门禁卡的时候，如果该门禁卡绑定了人脸图片信息，需要先删除人脸图片，再删除门禁卡信息。
- 在查询门禁卡信息过程中，首先，设备必须支持该功能，且该设备本身有门禁卡信息数据。

8.2.4 示例代码

8.2.4.1 门禁卡信息的查询

```
NET_OUT_FIND_RECORD_PARAM stuOutParam= sizeof(NET_OUT_FIND_RECORD_PARAM);
NET_IN_FIND_RECORD_PARAM stuInParam= sizeof(stuInParam);

stuInParam.emType = NET_RECORD_ACCESSCTLCARD;
FIND_RECORD_ACCESSCTLCARD_CONDITION *pStuCardInfo = NEW
FIND_RECORD_ACCESSCTLCARD_CONDITION;
pStuCardInfo->dwSize = sizeof(FIND_RECORD_ACCESSCTLCARD_CONDITION);
stuInParam.pQueryCondition = (void*)pStuCardInfo;

// 开始查询，设置查询条件
bRet = CLIENT_FindRecord(m_ILoginID, &stuInParam, &stuOutParam, DEFAULT_WAIT_TIME);
if (bRet == FALSE)
{
    printf("CLIENT_FindRecord fail \n");
}

if (pStuCardInfo)
```

```

{
    delete pStuCardInfo;
    pStuCardInfo = NULL;
}

// 查询 100 条数据
NET_IN_FIND_NEXT_RECORD_PARAM stuInParam= sizeof(stuInParam);
NET_OUT_FIND_NEXT_RECORD_PARAM stuOutParam = sizeof(stuOutParam);
stuInParam.lFindHandle = m_lFindHandle;
stuInParam.nFileCount = 100;
memset(pAccessCardInfo, 0, stuInParam.nFileCount * sizeof(NET_RECORDSET_ACCESS_CTL_CARD));

for (int i = 0; i < stuInParam.nFileCount; i++)
{
    pAccessCardInfo[i].dwSize = sizeof(NET_RECORDSET_ACCESS_CTL_CARD);
}
stuOutParam.pRecordList = (void*)pAccessCardInfo;
stuOutParam.nMaxRecordNum = stuInParam.nFileCount;

int nRet = CLIENT_FindNextRecord(&stuInParam, &stuOutParam, DEFAULT_WAIT_TIME);

if (!nRet)
{
    printf("CLIENT_ FindNextRecord failed \n");
}
// 结束查询
CLIENT_FindRecordClose(m_lFindHandle);

```

8.2.4.2 门禁卡信息的增加、修改、删除、清除

```

// 门禁卡信息的增加
NET_CTRL_RECORDSET_INSERT_PARAM stuInParam = sizeof(stuInParam);
stuInParam.stuCtrlRecordSetInfo.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_IN);
stuInParam.stuCtrlRecordSetResult.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_OUT);
stuInParam.stuCtrlRecordSetInfo.emType = NET_RECORD_ACCESSCTLCARD;

NET_RECORDSET_ACCESS_CTL_CARD *pStrCardInfo = NEW
NET_RECORDSET_ACCESS_CTL_CARD;

pStrCardInfo->dwSize = sizeof(NET_RECORDSET_ACCESS_CTL_CARD);

```

```

strncpy(pStrCardInfo->szCardNo, m_StuAddCardInfo.szCardNo, DH_MAX_CARDNO_LEN - 1);
strncpy(pStrCardInfo->szCardName, m_StuAddCardInfo.szCardName, DH_MAX_CARDNAME_LEN - 1);
strncpy(pStrCardInfo->szUserID, m_StuAddCardInfo.szUserID, DH_MAX_USERID_LEN - 1);
strncpy(pStrCardInfo->szPsw, m_StuAddCardInfo.szPsw, DH_MAX_CARDPWD_LEN - 1);
pStrCardInfo->emStatus = NET_ACCESSCTLCARD_STATE_NORMAL;
pStrCardInfo->emType = NET_ACCESSCTLCARD_TYPE_GENERAL;
pStrCardInfo->nUserTime = m_StuAddCardInfo.nUserTime;
pStrCardInfo->bFirstEnter = TRUE;
pStrCardInfo->bIsValid = TRUE;
pStrCardInfo->stuValidStartTime = m_StuAddCardInfo.stuValidStartTime;
pStrCardInfo->stuValidEndTime = m_StuAddCardInfo.stuValidEndTime;

// DoorNum 固定为 2，表示闸机的两扇门
pStrCardInfo->nDoorNum = 2;
pStrCardInfo->sznDoors[0] = 0;
pStrCardInfo->sznDoors[1] = 1;
// 控制两扇门的开门有效时间，255 表示全天
pStrCardInfo->nTimeSectionNum = 2;
pStrCardInfo->sznTimeSectionNo[0] = 255;
pStrCardInfo->sznTimeSectionNo[1] = 255;

stuInParam.stuCtrlRecordSetInfo.nBufLen = sizeof(NET_RECORDSET_ACCESS_CTL_CARD);
stuInParam.stuCtrlRecordSetInfo.pBuf = (void*)pStrCardInfo;

BOOL bRet = CLIENT_ControlDevice(m_ILoginID, DH_CTRL_RECORDSET_INSERT, &stuInParam,
DEFAULT_WAIT_TIME);
if (bRet == FALSE)
{
    Printf("CLIENT_ControlDevice insert card fail \n");
    return;
}
// 门禁卡信息的修改
NET_CTRL_RECORDSET_PARAM stuInParam = sizeof(stuInParam);
stuInParam.emType = NET_RECORD_ACCESSCTLCARD;

NET_RECORDSET_ACCESS_CTL_CARD *pStrCardInfo = NEW
NET_RECORDSET_ACCESS_CTL_CARD;
memset(pStrCardInfo, 0, sizeof(NET_RECORDSET_ACCESS_CTL_CARD));

pStrCardInfo->dwSize = sizeof(NET_RECORDSET_ACCESS_CTL_CARD);

```



```

strncpy(pStrCardInfo->szCardNo, m_CardInfo.szCardNo, DH_MAX_CARDNO_LEN - 1);
strncpy(pStrCardInfo->szCardName, m_CardInfo.szCardName, DH_MAX_CARDNAME_LEN - 1);
strncpy(pStrCardInfo->szUserID, m_CardInfo.szUserID, DH_MAX_USERID_LEN - 1);
strncpy(pStrCardInfo->szPsw, m_CardInfo.szPsw, DH_MAX_CARDPWD_LEN - 1);
pStrCardInfo->emStatus = NET_ACCESSCTLCARD_STATE_NORMAL;
pStrCardInfo->emType = NET_ACCESSCTLCARD_TYPE_GENERAL;
pStrCardInfo->nUserTime = m_CardInfo.nUserTime;
pStrCardInfo->bFirstEnter = TRUE;
pStrCardInfo->bIsValid = TRUE;
pStrCardInfo->stuValidStartTime = m_CardInfo.stuValidStartTime;
pStrCardInfo->stuValidEndTime = m_CardInfo.stuValidEndTime;
pStrCardInfo->nRecNo = m_CardInfo.nRecNo;

// DoorNum 固定为 2 表示闸机的两扇门
pStrCardInfo->nDoorNum = 2;
pStrCardInfo->sznDoors[0] = 0;
pStrCardInfo->sznDoors[1] = 1;
// 控制两扇门的开门有效时间 255 表示全天
pStrCardInfo->nTimeSectionNum = 2;
pStrCardInfo->sznTimeSectionNo[0] = 255;
pStrCardInfo->sznTimeSectionNo[1] = 255;

stuInParam.nBufLen = sizeof(NET_RECORDSET_ACCESS_CTL_CARD);
stuInParam.pBuf = (void*)pStrCardInfo;

BOOL bRet = CLIENT_ControlDevice(m_ILoginID, DH_CTRL_RECORDSET_UPDATEEX, &stuInParam,
DEFAULT_WAIT_TIME);
if (FALSE == bRet)
{
    printf("CLIENT_ControlDevice modify cardinfo fail");
    return;
}

// 门禁卡信息的删除
NET_CTRL_RECORDSET_PARAM stuInParam = sizeof(stuInParam);
stuInParam.emType = NET_RECORD_ACCESSCTLCARD;
stuInParam.pBuf = &pCardInfo->nRecNo;
stuInParam.nBufLen = sizeof(int);
BOOL bRet = CLIENT_ControlDevice(m_ILoginID, DH_CTRL_RECORDSET_REMOVE, &stuInParam,
DEFAULT_WAIT_TIME);

```

```
if (FALSE == bRet)
{
    printf(("CLIENT_ControlDevice   delete cardinfo fail\n"));
    return;
}

// 门禁卡信息的清除(删除所有门禁卡信息)
NET_CTRL_RECORDSET_PARAM stuInParam = sizeof(stuInParam);
stuInParam.emType = NET_RECORD_ACCESSCTLCARD;

BOOL  bRet  =  CLIENT_ControlDevice(m_ILoginID,  DH_CTRL_RECORDSET_CLEAR,  &stuInParam,
DEFAULT_WAIT_TIME);
if (FALSE == bRet)
{
    printf ("CLIENT_ControlDevice   clear cardinfo fail\n");
    return;
}
```

8.3 人脸管理

8.3.1 简介

人脸图片的增加、修改、删除、清除，一般的使用场景和人员信息配套起来使用。

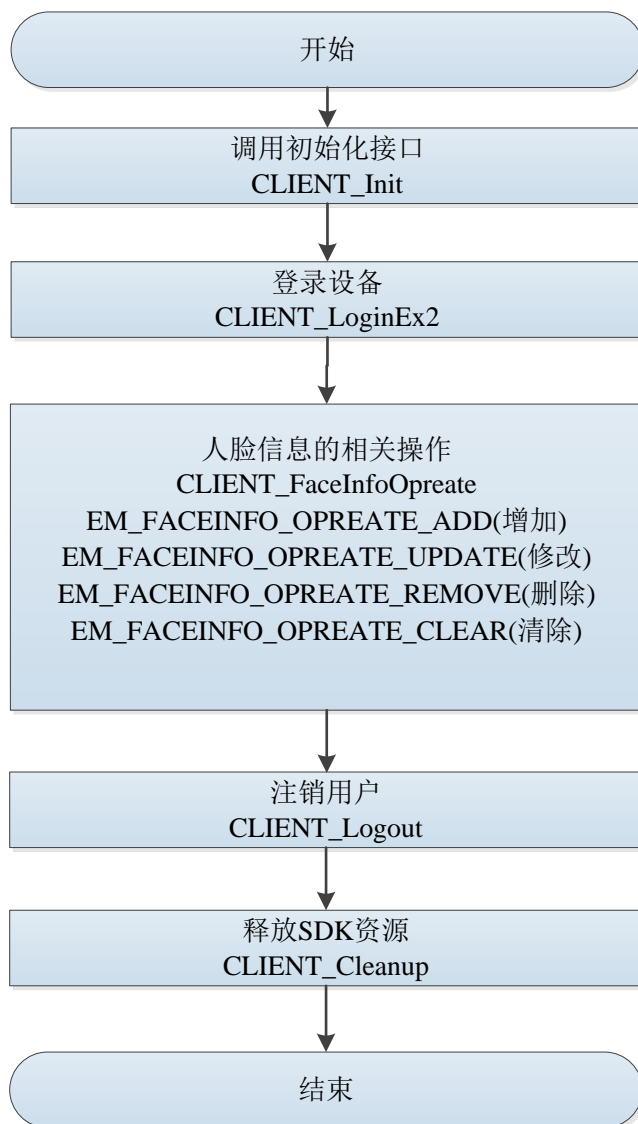
8.3.2 接口总览

接口	说明
CLIENT_FaceInfoOpreate	人脸图片信息的增加、修改、删除、清除
CLIENT_FindRecord	设置查询条件
CLIENT_FindNextRecord	在当前查询条件下查询数据
CLIENT_FindRecordClose	关闭查询
CLIENT_ControlDevice	人员信息的增加、修改、删除、清除

8.3.3 流程说明

人脸图片信息的增删改和清除流程，如图 8-2 所示。

图8-2 人脸图片信息的增删改和清除



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用 CLIENT_FaceInfoOpreate，根据相应的枚举类型进行人脸的增加、修改、删除和清除操作。
- 步骤4 业务使用完后，调用 CLIENT_Logout 登出设备。
- 步骤5 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- 人脸信息添加：使用的枚举为 EM_FACEINFO_OPREATE_ADD。
- 人脸信息修改：使用的枚举为 EM_FACEINFO_OPREATE_UPDATE。
- 人脸信息删除：使用的枚举为 EM_FACEINFO_OPREATE_REMOVE。
- 人脸信息清除：使用的枚举为 EM_FACEINFO_OPREATE_CLEAR。
- 人脸信息的修改和删除都是根据门禁卡的 UserID 进行操作的。

8.3.4 示例代码

```
// 人脸图片信息的增加
NET_IN_ADD_FACE_INFO stuInParam = sizeof(NET_IN_ADD_FACE_INFO);
strncpy(stuInParam.szUserID, m_StuAddCardInfo.szUserID, DH_MAX_USERID_LEN - 1);
stuInParam.stuFaceInfo.nFacePhoto = 1;

FILE *fPic = fopen(m_szFilePath, "rb");
fseek(fPic, 0, SEEK_END);
int nLength = ftell(fPic);
rewind(fPic);

stuInParam.stuFaceInfo.nFacePhotoLen[0] = nLength;
stuInParam.stuFaceInfo.pszFacePhoto[0] = new char[nLength];

memset(stuInParam.stuFaceInfo.pszFacePhoto[0], 0, nLength);
int nReadLen = fread(stuInParam.stuFaceInfo.pszFacePhoto[0], 1, nLength, fPic);
fclose(fPic);
fPic = NULL;

NET_OUT_ADD_FACE_INFO stuOutParam = sizeof(stuOutParam);

BOOL bRet = CLIENT_FaceInfoOpreate(m_lLoginID, EM_FACEINFO_OPREATE_ADD, (void*)&stuInParam,
(void*)&stuOutParam, DEFAULT_WAIT_TIME);
if (bRet == FALSE)
{
    printf("CLIENT_FaceInfoOpreate add face fail");
}

if (fPic)
{
    fclose(fPic);
    fPic = NULL;
}
if (stuInParam.stuFaceInfo.pszFacePhoto[0])
{
    delete[] stuInParam.stuFaceInfo.pszFacePhoto[0];
    stuInParam.stuFaceInfo.pszFacePhoto[0] = NULL;
}

// 人脸图片信息的修改
```

```

NET_IN_UPDATE_FACE_INFO stuInParam = sizeof(stuInParam);
strncpy(stuInParam.szUserID, m_CardInfo.szUserID, sizeof(m_CardInfo.szUserID) - 1);
stuInParam.stuFaceInfo.nFacePhoto = 1;

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

fseek(fPic, 0, SEEK_END);
int nLength = ftell(fPic);
rewind(fPic);

stuInParam.stuFaceInfo.nFacePhotoLen[0] = nLength;
stuInParam.stuFaceInfo.pszFacePhoto[0] = new char[nLength];

memset(stuInParam.stuFaceInfo.pszFacePhoto[0], 0, nLength);
int nReadLen = fread(stuInParam.stuFaceInfo.pszFacePhoto[0], 1, nLength, fPic);
fclose(fPic);
fPic = NULL;

NET_OUT_UPDATE_FACE_INFO stuOutParam;
memset(&stuOutParam, 0, sizeof(stuOutParam));
stuOutParam.dwSize = sizeof(stuOutParam);

BOOL    bRet    =    CLIENT_FaceInfoOpreate(mILoginID,    EM_FACEINFO_OPREATE_UPDATE,
(void*)&stuInParam, (void*)&stuOutParam, DEFAULT_WAIT_TIME);
if (bRet == FALSE)
{
    printf ("CLIENT_FaceInfoOpreate modify face fail");
}

if (fPic)
{
    fclose(fPic);
    fPic = NULL;
}
if (stuInParam.stuFaceInfo.pszFacePhoto[0])
{
    delete[] stuInParam.stuFaceInfo.pszFacePhoto[0];
    stuInParam.stuFaceInfo.pszFacePhoto[0] = NULL;
}

```

```

// 人脸图片的删除
NET_IN_REMOVE_FACE_INFO stuInParam = sizeof(stuInParam);
//通过门禁卡信息的 UserID 来删除对应的人脸图片
strncpy(stuInParam.szUserID, cardInfo.szUserID, DH_MAX_USERID_LEN - 1);

NET_OUT_REMOVE_FACE_INFO stuOutParam;
memset(&stuOutParam, 0, sizeof(stuOutParam));
stuOutParam.dwSize = sizeof(stuOutParam);

bRet = CLIENT_FaceInfoOpreate(m_lLoginID, EM_FACEINFO_OPREATE_REMOVE, (void*)&stuInParam,
(void*)&stuOutParam, DEFAULT_WAIT_TIME);
if (bRet == FALSE)
{
    printf ("CLIENT_FaceInfoOpreate delete face fail");
}

// 人脸图片信息的清除(删除所有的人脸图片信息)
NET_IN_CLEAR_FACE_INFO stuInParam = sizeof(stuInParam);
NET_OUT_CLEAR_FACE_INFO stuOutParam;
memset(&stuOutParam, 0, sizeof(stuOutParam));
stuOutParam.dwSize = sizeof(stuOutParam);

BOOL    bRet    =    CLIENT_FaceInfoOpreate(m_lLoginID,    EM_FACEINFO_OPREATE_CLEAR,
(void*)&stuInParam, (void*)&stuOutParam, DEFAULT_WAIT_TIME);
if (bRet == FALSE)
{
    printf ("CLIENT_FaceInfoOpreate clear face fail");
}

```

8.4 门禁进出记录查询

8.4.1 简介

门禁记录集的查询，查询信息包括卡编号、用户序号、开门状态、卡类型、开门方式、时间等信息。

8.4.2 接口总览

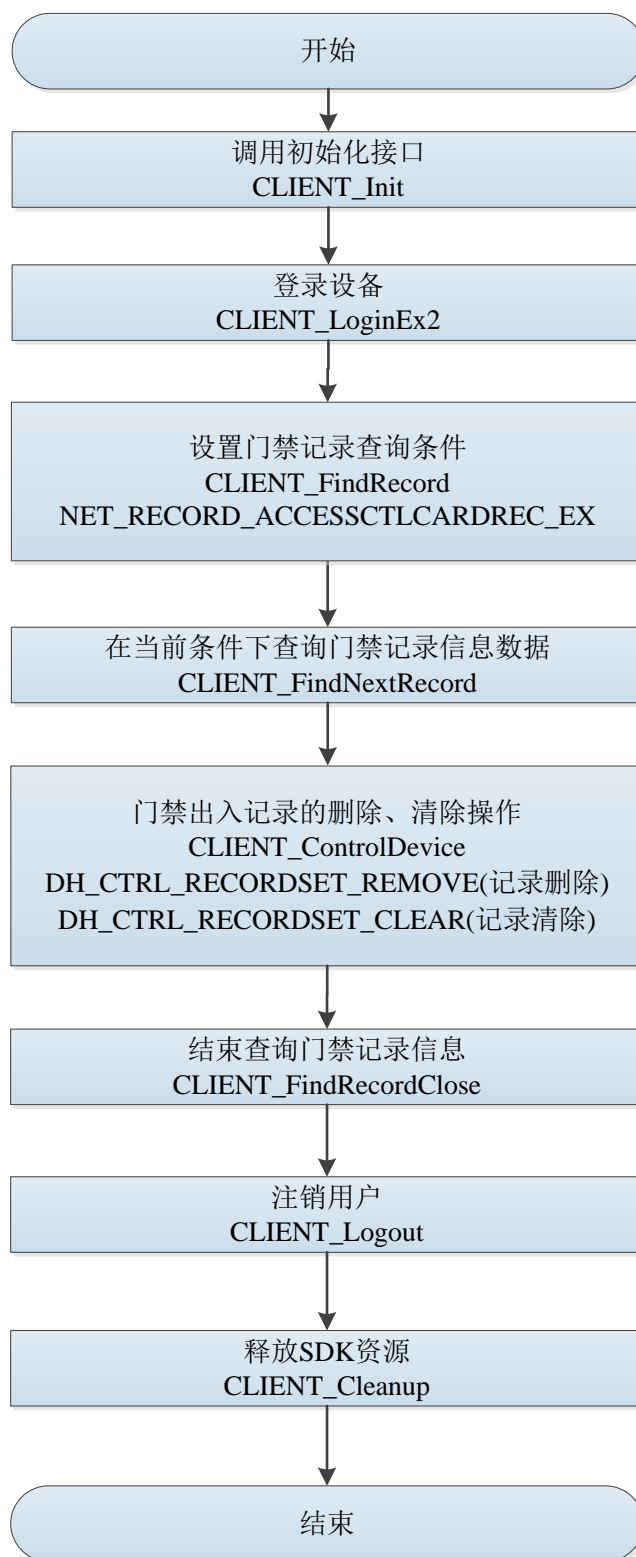
接口	说明
----	----

接口	说明
CLIENT_FindRecord	设置查询条件
CLIENT_FindNextRecord	在当前查询条件下查询数据
CLIENT_FindRecordClose	关闭查询
CLIENT_ControlDevice	门禁记录的删除、清除

8.4.3 流程说明

门禁记录的查找/删除/清除流程，如图 8-3 所示。

图8-3 门禁记录的查找/删除/清除



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginEx2 登录设备。
- 步骤3 调用 CLIENT_FindRecord 设置门禁记录的查询条件。

- 步骤4 调用 `CLIENT_FindNextRecord`，在当前指定条件下查找固定数量的门禁记录信息。
- 步骤5 调用 `CLIENT_ControlDevice`，其中枚举类型为 `DH_CTRL_RECORDSET_REMOVE`（删除）和 `DH_CTRL_RECORDSET_CLEAR`（清除）进行门禁记录的删除和清除操作。
- 步骤6 查询结束后，调用 `CLIENT_FindRecordClose`，清除查询资源信息。
- 步骤7 业务使用完后，调用 `CLIENT_Logout` 登出设备。
- 步骤8 SDK 功能使用完后，调用 `CLIENT_Cleanup` 释放 SDK 资源。

注意事项

- 门禁记录的删除：使用的枚举为 `DH_CTRL_RECORDSET_REMOVE`。
- 门禁记录的清除：使用的枚举为 `DH_CTRL_RECORDSET_CLEAR`。

8.4.4 示例代码

查询实例代码请参见“8.2.4.1 门禁卡信息的查询”的相关示例代码。

门禁记录的删除和清除请参见“8.2.4.2 门禁卡信息的增加、修改、删除、清除”中相关删除和清除的示例代码。

9.1 SDK 初始化

9.1.1 SDK 初始化 CLIENT_Init

选项	说明	
描述	对整个 SDK 进行初始化	
函数	<pre> BOOL CLIENT_Init(fDisconnect cbDisconnect, LDWORD dwUser); </pre>	
参数	[in]cbDisconnect	断线回调函数
	[in]dwUser	断线回调函数的用户参数
返回值	成功返回 TRUE，失败返回 FALSE	
说明	调用网络 SDK 其他函数的前提 回调函数设置成 NULL 时，设备断线后不会回调给用户	

9.1.2 SDK 清理 CLIENT_Cleanup

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

9.1.3 设置断线重连回调函数 CLIENT_SetAutoReconnect

选项	说明	
描述	设置自动重连回调函数	
函数	<pre> void CLIENT_SetAutoReconnect(fHaveReConnect cbAutoConnect, LDWORD dwUser); </pre>	
参数	[in]cbAutoConnect	断线重连回调函数
	[in]dwUser	断线重连回调函数的用户参数
返回值	无	
说明	设置断线重连回调接口。如果回调函数设置为 NULL，则不自动重连	

9.1.4 设置网络参数 CLIENT_SetNetworkParam

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

9.2 设备登录

9.2.1 用户登录设备 CLIENT_LoginEx2

选项	说明	
描述	用户登录设备	
函数	LLONG CLIENT_LoginEx2(const char *pchDVRIP, WORD wDVRPort, const char *pchUserName, const char *pchPassword, EM_LOGIN_SPAC_CAP_TYPE emSpecCap, void* pCapParam, LPNET_DEVICEINFO_Ex lpDeviceInfo, int *error);	
参数	[in]pchDVRIP	设备 IP
	[in]wDVRPort	设备端口
	[in]pchUserName	用户名
	[in]pchPassword	密码
	[in]emSpecCap	登录类别
	[in]pCapParam	登录类别参数
	[out]lpDeviceInfo	设备信息
	[out]error	失败的错误码
返回值	成功返回非 0，失败返回 0	
说明	无	

参数 error 的错误码及含义说明，请参见表 9-1。

表9-1 参数 error 的错误码及含义

error 的错误码	对应的含义
1	密码不正确
2	用户名不存在
3	登录超时

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

9.2.2 用户登出设备 CLIENT_Logout

选项	说明	
描述	用户登出设备	
函数	<pre> BOOL CLIENT_Logout(LLONG ILoginID); </pre>	
参数	[in]ILoginID	CLIENT_LoginEx2 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.3 实时监控

9.3.1 打开监视 CLIENT_RealPlayEx

选项	说明	
描述	打开实时监控	
函数	<pre> LLONG CLIENT_RealPlayEx(LLONG ILoginID, int nChannelID, HWND hWnd, DH_RealPlayType rType); </pre>	
参数	[in]ILoginID	CLIENT_LoginEx2 的返回值
	[in]nChannelID	视频通道号，从 0 开始递增的整数
	[in]hWnd	窗口句柄，仅在 Windows 系统下有效
	[in]rType	预览类型
返回值	成功返回非 0，失败返回 0	

选项	说明
说明	在 Windows 环境下： hWnd 为有效值时，在对应窗口显示画面 hWnd 为 NULL 时，表示取流方式，通过设置回调函数来获取视频数据，交由用户处理

预览类型及含义请参见表 9-2。

表9-2 预览类型说明

预览类型	含义
DH_RType_Realplay	实时预览
DH_RType_Multiplay	多画面预览
DH_RType_Realplay_0	实时监视-主码流，等同于 DH_RType_Realplay
DH_RType_Realplay_1	实时监视-从码流 1
DH_RType_Realplay_2	实时监视-从码流 2
DH_RType_Realplay_3	实时监视-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

9.3.2 关闭监视 CLIENT_StopRealPlayEx

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

9.3.3 保存监视数据 CLIENT_SaveRealData

选项	说明
描述	保存实时监视数据为文件
函数	BOOL CLIENT_SaveRealData(LLONG IRealHandle, const char *pchFileName);
参数	[in] IRealHandle CLIENT_RealPlayEx 的返回值

选项	说明	
	[in] pchFileName	需要保存的文件路径
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.3.4 停止保存监视数据 CLIENT_StopSaveRealData

选项	说明	
描述	停止保存实时监视数据为文件	
函数	<pre> BOOL CLIENT_StopSaveRealData(LONG IRealHandle); </pre>	
参数	[in] IRealHandle	CLIENT_RealPlayEx 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.3.5 设置监视数据回调 CLIENT_SetRealDataCallBackEx

选项	说明	
描述	设置实时监视数据回调	
函数	<pre> BOOL CLIENT_SetRealDataCallBackEx(LONG IRealHandle, fRealDataCallBackEx cbRealData, LDWORD dwUser, DWORD dwFlag); </pre>	
参数	[in] IRealHandle	CLIENT_RealPlayEx 的返回值
	[in] cbRealData	监视数据流回调函数
	[in] dwUser	监视数据流回调函数的参数
	[in] dwFlag	回调中监视数据的类型
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

表9-3 dwFlag 类型及含义

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

9.4 智能事件订阅

9.4.1 智能事件订阅 CLIENT_RealLoadPictureEx

选项	说明
----	----

选项	说明	
描述	订阅智能事件接口	
函数	<pre>LLONG CLIENT_RealLoadPictureEx(LLONG ILoginID, int nChannelID, DWORD dwAlarmType, BOOL bNeedPicFile, fAnalyzerDataCallBack cbAnalyzerData, LDWORD dwUser, void* Reserved);</pre>	
参数	[in] ILoginID	登录句柄
	[in] nChannelID	通道号
	[in] dwAlarmType	报警类型
	[in] bNeedPicFile	是否订阅图片文件，1-订阅图片文件，在回调函数中会返回智能图片信息，0-不订阅图片文件，在回调函数中不会返回智能图片信息（在不需要图片信息时，可减少网络流量）
	[in] cbAnalyzerData	智能数据分析回调函数
	[in] dwUser	用户参数
	[in] Reserved	保留参数
返回值	成功返回 LLONG 类型的订阅句柄，失败返回 0	
说明	接口返回失败，请用 CLIENT_GetLastError 获取错误码	

本手册使用到的智能事件请参见下表。

预览类型	含义
EVENT_IVS_FACEDETECT	人脸检测
EVENT_IVS_FACERECOGNITION	人脸识别
EVENT_IVS_TRAFFICJUNCTION	交通路口事件
EVENT_IVS_TRAFFIC_PEDESTRAIN	行人事件
EVENT_IVS_TRAFFICJAM	交通拥堵事件
EVENT_IVS_TRAFFIC_OVERSPEED	超速事件
EVENT_IVS_TRAFFIC_UNDERSPEED	欠速事件
EVENT_IVS_TRAFFIC_FLOWSTATE	车流量事件
EVENT_IVS_HUMANTRAIT	人体检测事件
EVENT_IVS_CROSSLINEDETECTION	绊线入侵事件
EVENT_IVS_CROSSREGIONDETECTION	区域入侵事件
EVENT_IVS_NUMBERSTAT	人数统计事件
EVENT_IVS_MAN_NUM_DETECTION	区域内人数统计事件
EVENT_IVS_ACCESS_CTL	门禁事件

9.4.2 智能事件的取消订阅 CLIENT_StopLoadPic

选项	说明
描述	停止智能事件订阅

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

9.5 智能事相关录像图片查询与下载

9.5.1 按条件查询图片或录像 CLIENT_FindFileEx

选项	说明	
描述	按条件查找文件	
函数	LLONG CLIENT_FindFileEx(LLONG lLoginID, EM_FILE_QUERY_TYPE emType, void* pQueryCondition, void* reserved, int waittime);	
参数	[in] lLoginID	登录句柄
	[in] emType	查询的文件类型
	[in] pQueryCondition	查询条件
	[in] reserved	保留字节
	[in] waittime	等待时间
返回值	成功返回 LLONG 类型的查询句柄，失败返回 0	
说明	无	

9.5.2 获取查询到文件个数 CLIENT_GetTotalFileCount

选项	说明	
描述	获取查询文件的个数	
函数	BOOL CLIENT_GetTotalFileCount(LLONG lFindHandle, int* pTotalCount, void * reserved, int waittime);	
参数	[in] lFindHandle	查询句柄
	[out] pTotalCount	查询到的数目
	[in] reserved	保留字节
	[in] waittime	超时时间

选项	说明
返回值	成功返回 TRUE，失败返回 FALSE
说明	无

9.5.3 查找文件 CLIENT_FindNextFileEx

选项	说明	
描述	查找文件	
函数	<pre>int CLIENT_FindNextFileEx(LLONG IFindHandle, int nFilecount, void* pMediaFileInfo, int maxlen, void* reserved, int waittime);</pre>	
参数	[in] IFindHandle	查询句柄
	[in] nFilecount	需要查询文件的数目
	[out] pMediaFileInfo	文件缓冲区，为查找长度的数组首地址
	[in] maxlen	查找文件数组缓冲区大小
	[in] reserved	保留字节
	[in] waittime	超时时间
返回值	成功返回查找的文件个数，失败返回-1，返回 0 表示查找结束	
说明	无	

9.5.4 结束文件查找 CLIENT_FindCloseEx

选项	说明	
描述	结束文件查找	
函数	<pre>BOOL CLIENT_FindCloseEx(LLONG IFindHandle);</pre>	
参数	[in] IFindHandle	查询句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.5.5 开始录像回放 CLIENT_PlayBackByTimeEx2

选项	说明
描述	开始录像回放

选项	说明	
函数	<pre> LLONG CLIENT_PlayBackByTimeEx2(LLONG ILoginID, Int nChannelID, NET_IN_PLAY_BACK_BY_TIME_INFO* pstNetIn, NET_OUT_PLAY_BACK_BY_TIME_INFO* pstNetOut); </pre>	
参数	[in] ILoginID	登录句柄
	[in] nChannelID	通道号
	[in] pstNetIn	回放输入参数
	[out] pstNetOut	回放输出参数
返回值	成功返回 LLONG 类型的回放句柄，失败返回 0	
说明	无	

9.5.6 结束录像回放 CLIENT_StopPlayBack

选项	说明	
描述	结束录像回放	
函数	<pre> BOOL CLIENT_StopPlayBack(LLONG IPlayHandle); </pre>	
参数	[in] IPlayHandle	回放句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.5.7 开始录像下载 CLIENT_DownloadByTimeEx

选项	说明	
描述	开始录像下载	
函数	<pre> LLONG CLIENT_PlayBackByTimeEx2(LLONG ILoginID, int nChannelID, NET_IN_PLAY_BACK_BY_TIME_INFO* pstNetIn, NET_OUT_PLAY_BACK_BY_TIME_INFO* pstNetOut); </pre>	
参数	[in] ILoginID	登录句柄
	[in] nChannelID	通道号
	[in] pstNetIn	回放输入参数
	[out] pstNetOut	回放输出参数
返回值	成功返回 LLONG 类型的回放句柄，失败返回 0	
说明	无	

9.5.8 停止录像下载 CLIENT_StopDownload

选项	说明
----	----

选项	说明	
描述	停止录像下载	
函数	<pre> BOOL CLIENT_StopDownload(LLONG IFileHandle); </pre>	
参数	[in] IFileHandle	回放句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.5.9 下载图片 CLIENT_DownloadRemoteFile

选项	说明	
描述	通过文件名下载文件	
函数	<pre> BOOL CLIENT_DownloadRemoteFile(LLONG ILoginID, const DH_IN_DOWNLOAD_REMOTE_FILE* pInParam, DH_OUT_DOWNLOAD_REMOTE_FILE* pOutParam, int nWaitTime); </pre>	
参数	[in] IFileHandle	回放句柄
	[in] pInParam	下载文件输入参数
	[out] pOutParam	下载文件输出参数
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.6 人脸事件订阅

人脸事件订阅，请参见“9.4 智能事件订阅”。

9.7 人脸库的增删改查

9.7.1 人脸库的增删改操作 CLIENT_OperateFaceRecognitionGroup

选项	说明
描述	人脸库的增删改操作

选项	说明	
函数	BOOL CLIENT_OperateFaceRecognitionGroup(LLONG ILoginID, const NET_IN_OPERATE_FACERECONGNITION_GROUP* pstInParam, NET_OUT_OPERATE_FACERECONGNITION_GROUP *pstOutParam, int nWaitTime);	
参数	[in] ILoginID	登录句柄
	[in] pInParam	输入参数
	[out] pstOutParam	输出参数
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.7.2 人脸库信息查询 CLIENT_FindGroupInfo

选项	说明	
描述	人脸库的查询操作	
函数	BOOL CLIENT_FindGroupInfo(LLONG ILoginID, const NET_IN_FIND_GROUP_INFO* pstInParam, NET_OUT_FIND_GROUP_INFO *pstOutParam, int nWaitTime);	
参数	[in] ILoginID	登录句柄
	[in] pInParam	输入参数
	[out] pstOutParam	输出参数
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.8 人脸的增删改查

9.8.1 人脸的增删改操作 CLIENT_OperateFaceRecognitionDB

选项	说明	
描述	人脸的增删改操作	
函数	BOOL CLIENT_OperateFaceRecognitionDB(LLONG ILoginID, const NET_IN_OPERATE_FACERECONGNITIONDB* pstInParam, NET_OUT_OPERATE_FACERECONGNITIONDB *pstOutParam, Int nWaitTime);	

选项	说明	
参数	[in] ILoginID	登录句柄
	[in] pInParam	输入参数
	[out] pstOutParam	输出参数
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.8.2 设置人脸的查询条件 CLIENT_OperateFaceRecognitionDB

选项	说明	
描述	设置人脸的查询条件	
函数	<pre> BOOL CLIENT_StartFindFaceRecognition(LLONG ILoginID, const NET_IN_STARTFIND_FACERECONGNITION* pstInParam, NET_OUT_STARTFIND_FACERECONGNITION *pstOutParam, int nWaitTime); </pre>	
参数	[in] ILoginID	登录句柄
	[in] pInParam	输入参数
	[out] pstOutParam	输出参数
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.8.3 查询人脸 CLIENT_DoFindFaceRecognition

选项	说明	
描述	查询人脸	
函数	<pre> BOOL CLIENT_DoFindFaceRecognitionRecord(const NET_IN_DOFIND_FACERECONGNITIONRECORD* pstInParam, NET_OUT_DOFIND_FACERECONGNITIONRECORD *pstOutParam, int nWaitTime); </pre>	
参数	[in] pInParam	输入参数
	[out] pstOutParam	输出参数
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.8.4 结束人脸查找 CLIENT_StopFindFaceRecognition

选项	说明
描述	结束人脸查找

选项	说明	
函数	BOOL CLIENT_StopFindFaceRecognition(LLONG IFindHandle);	
参数	[in] IFindHandle	查询句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.8.5 以库为对象进行布控 CLIENT_FaceRecognitionPutDisposition

选项	说明	
描述	以库为对象进行布控	
函数	BOOL CLIENT_FaceRecognitionPutDisposition(LLONG ILoginID, const NET_IN_FACE_RECOGNITION_PUT_DISPOSITION_INFO* pstInParam, NET_OUT_FACE_RECOGNITION_PUT_DISPOSITION_INFO *pstOutParam, int nWaitTime);	
参数	[in] ILoginID	登录句柄
	[in] pInParam	输入参数
	[out] pstOutParam	输出参数
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.8.6 以库为对象进行撤控 CLIENT_FaceRecognitionDelDisposition

选项	说明	
描述	以库为对象进行撤控	
函数	BOOL CLIENT_FaceRecognitionDelDisposition(LLONG ILoginID, const NET_IN_FACE_RECOGNITION_DEL_DISPOSITION_INFO* pstInParam, NET_OUT_FACE_RECOGNITION_DEL_DISPOSITION_INFO *pstOutParam, int nWaitTime);	
参数	[in] ILoginID	登录句柄
	[in] pInParam	输入参数
	[out] pstOutParam	输出参数
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.8.7 以通道为对象进行布控 CLIENT_SetGroupInfoForChannel

选项	说明	
描述	以通道为对象进行布控	
函数	<pre> BOOL CLIENT_SetGroupInfoForChannel(LLONG ILoginID, const NET_IN_SET_GROUPINFO_FOR_CHANNEL* pstInParam, NET_OUT_SET_GROUPINFO_FOR_CHANNEL *pstOutParam, int WaitTime); </pre>	
参数	[in] ILoginID	登录句柄
	[in] pInParam	输入参数
	[out] pstOutParam	输出参数
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.8.8 订阅人脸查询进度 CLIENT_AttachFaceFindState

选项	说明	
描述	订阅人脸查询进度	
函数	<pre> LLONG CLIENT_AttachFaceFindState(LLONG ILoginID, const NET_IN_FACE_FIND_STATE* pstInParam, NET_OUT_FACE_FIND_STATE *pstOutParam, Int nWaitTime); </pre>	
参数	[in] ILoginID	登录句柄
	[in] pInParam	输入参数
	[out] pstOutParam	输出参数
	[in] nWaitTime	超时时间
返回值	成功返回人脸进度句柄，失败返回 0	
说明	无	

9.8.9 取消订阅人脸查询进度 CLIENT_DetachFaceFindState

选项	说明	
描述	取消订阅人脸查询进度	
函数	<pre> BOOL CLIENT_DetachFaceFindState(LLONG IAttachHandle); </pre>	
参数	[in] IAttachHandle	CLIENT_AttachFaceFindState 返回的句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.9 人体检测

9.9.1 人体图片下载 CLIENT_DownloadRemoteFile

选项	说明	
描述	人体图片下载	
函数	BOOL CLIENT_DownloadRemoteFile(LLONG ILoginID, const DH_IN_DOWNLOAD_REMOTE_FILE* pInParam, DH_OUT_DOWNLOAD_REMOTE_FILE* pOutParam, int nWaitTime = 1000);	
参数	[in] ILoginID	CLIENT_AttachFaceFindState 返回的句柄
	[in] pInParam	输入参数
	[out] pOutParam	输出参数
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.10 客流量统计

9.10.1 客流量事件订阅 CLIENT_AttachVideoStatSummary

选项	说明	
描述	客流量事件订阅	
函数	LLONG CLIENT_AttachVideoStatSummary(LLONG ILoginID, const NET_IN_ATTACH_VIDEOSTAT_SUM* pInParam, NET_OUT_ATTACH_VIDEOSTAT_SUM* pOutParam, int nWaitTime);	
参数	[in] ILoginID	登录句柄
	[in] pInParam	订阅客流输入参数
	[out] pOutParam	订阅客流输出参数
	[in] nWaitTime	超时时间
返回值	客流订阅句柄	
说明	无	

9.10.2 取消订阅客流量事件 CLIENT_DetachVideoStatSummary

选项	说明
描述	取消订阅客流量事件

选项	说明	
函数	BOOL CLIENT_DetachVideoStatSummary(LLONG IAttachHandle);	
参数	[in] IAttachHandle	客流订阅句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.10.3 开始查询客流历史数据（设置查询条件）

CLIENT_StartFindNumberStat

选项	说明	
描述	开始查询客流历史数据（设置查询条件）	
函数	LLONG CLIENT_StartFindNumberStat(LLONG ILoginID, NET_IN_FINDNUMBERSTAT* pstInParam, NET_OUT_FINDNUMBERSTAT* pstOutParam);	
参数	[in] ILoginID	登录句柄
	[in] pstInParam	输入查询条件
	[out] pstOutParam	输出查询结果
返回值	查询句柄	
说明	无	

9.10.4 查询客流历史数据 CLIENT_DoFindNumberStat

选项	说明	
描述	开始查询客流历史数据（设置查询条件）	
函数	int CLIENT_DoFindNumberStat(LLONG IFindHandle, NET_IN_DOFINDNUMBERSTAT* pstInParam, NET_OUT_DOFINDNUMBERSTAT* pstOutParam);	
参数	[in] ILoginID	登录句柄
	[in] pstInParam	查询输入参数
	[out] pstOutParam	查询输出参数
返回值	查询数量	
说明	无	

9.10.5 结束查询历史数据 CLIENT_StopFindNumberStat

选项	说明	
描述	结束查询历史数据	

选项	说明	
函数	BOOL CLIENT_StopFindNumberStat(LLONG IFindHandle);	
参数	[in] IFindHandle	查询句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.11 智能交通

9.11.1 开始查找数据（设置查询条件）CLIENT_FindRecord

选项	说明	
描述	开始查找数据（设置查询条件）	
函数	BOOL CLIENT_FindRecord(LLONG ILoginID, NET_IN_FIND_RECORD_PARAM* pInParam, NET_OUT_FIND_RECORD_PARAM* pOutParam, int waittime=1000);	
参数	[in] ILoginID	登录句柄
	[in] pInParam	输入查询条件
	[out] pOutParam	输出查询结果
返回值	查询句柄	
说明	无	

9.11.2 查询数据总数 CLIENT_QueryRecordCount

选项	说明	
描述	查询数据总数	
函数	BOOL CLIENT_QueryRecordCount(NET_IN_QUEYT_RECORD_COUNT_PARAM* pInParam, NET_OUT_QUEYT_RECORD_COUNT_PARAM* pOutParam, int waittime=1000);	
参数	[in] pInParam	查询输入参数
	[out] pOutParam	查询输出参数
	[in] waittime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.11.3 查询指定条数数据 CLIENT_FindNextRecord

选项	说明	
描述	查询指定条数数据	
函数	int CLIENT_FindNextRecord(NET_IN_FIND_NEXT_RECORD_PARAM* pInParam, NET_OUT_FIND_NEXT_RECORD_PARAM* pOutParam, int waittime=1000);	
参数	[in] pstInParam	查询输入参数
	[out] pstOutParam	查询输出参数
	[in] waittime	超时时间
返回值	查询数量	
说明	无	

9.11.4 结束车流量查询 CLIENT_FindRecordClose

选项	说明	
描述	结束车流量查询	
函数	BOOL CLIENT_FindRecordClose(LLONG IFindHandle);	
参数	[in] IFindHandle	查询句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.11.5 黑白名单的增删改 CLIENT_OperateTrafficList

选项	说明	
描述	黑白名单的增删改	
函数	BOOL CLIENT_OperateTrafficList(LLONG ILoginID , NET_IN_OPERATE_TRAFFIC_LIST_RECORD* pstInParam , NET_OUT_OPERATE_TRAFFIC_LIST_RECORD *pstOutParam , int waittime));	
参数	[in] ILoginID	登录句柄
	[in] pstInParam	黑白名单操作输入参数
	[out] pstOutParam	黑白名单操作输出参数
	[in] waittime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	NET_TRAFFIC_LIST_INSERT// 增加记录操作 NET_TRAFFIC_LIST_UPDATE// 更新记录操作 NET_TRAFFIC_LIST_REMOVE// 删除记录操作	

9.11.6 批量下载文件 CLIENT_DownloadMultiFile

选项	说明	
描述	批量下载文件	
函数	BOOL CLIENT_DownloadMultiFile(LLONG ILoginID, NET_IN_DOWNLOAD_MULTI_FILE *pstInParam, NET_OUT_DOWNLOAD_MULTI_FILE *pstOutParam, int waittime=1000);	
参数	[in] ILoginID	登录句柄
	[in] pstInParam	查询输入参数
	[out] pstOutParam	查询输出参数
	[in] waittime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.11.7 停止批量下载文件 CLIENT_StopLoadMultiFile

选项	说明	
描述	停止批量下载文件	
函数	BOOL CLIENT_StopLoadMultiFile(LLONG IDownLoadHandle);	
参数	[in] IDownLoadHandle	批量下载句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

9.12 门禁

9.12.1 开始查找数据（设置查询条件） CLIENT_FindRecord

选项	说明	
描述	开始查找数据（设置查询条件）	
函数	BOOL CLIENT_FindRecord(LLONG ILoginID, NET_IN_FIND_RECORD_PARAM* pInParam, NET_OUT_FIND_RECORD_PARAM* pOutParam, int waittime=1000);	
参数	[in] ILoginID	登录句柄
	[int] pInParam	输入查询条件
	[out] pOutParam	输出查询结果

选项	说明
返回值	查询句柄
说明	无

9.12.2 查询指定条数数据 CLIENT_FindNextRecord

选项	说明
描述	查询指定条数数据
函数	<pre>int CLIENT_FindNextRecord(NET_IN_FIND_NEXT_RECORD_PARAM* pInParam, NET_OUT_FIND_NEXT_RECORD_PARAM* pOutParam, int waittime=1000);</pre>
参数	[int] pstInParam 查询输入参数
	[out] pstOutParam 查询输出参数
	[in] waittime 超时时间
返回值	查询数量
说明	无

9.12.3 结束查询 CLIENT_FindRecordClose

选项	说明
描述	结束查询
函数	<pre>BOOL CLIENT_FindRecordClose(LLONG IFindHandle);</pre>
参数	[in] IFindHandle 查询句柄
返回值	成功返回 TRUE，失败返回 FALSE
说明	无

9.12.4 人员/门禁记录信息操作 CLIENT_FindRecordClose

选项	说明
描述	人员信息的增删改查和清除，以及门禁记录的删除和清除
函数	<pre>BOOL CLIENT_ControlDevice(LLONG ILoginID , CtrlType type , void *param , int waittime = 1000);</pre>
参数	[in] ILoginID 登录句柄
	[in] type 控制类型
	[in] param 控制参数，根据 type 不同而不同
	[in] waittime 超时时间
返回值	成功返回 TRUE，失败返回 FALSE

选项	说明
说明	无

9.12.5 人脸图片的操作 CLIENT_FindRecordClose

选项	说明	
描述	人脸图片信息的增加、修改、删除和清除	
函数	<pre> BOOL CLIENT_FaceInfoOpreate(LLONG lLoginID, EM_FACEINFO_OPREATE_TYPE emType, void* pInParam, void* pOutParam, int nWaitTime = 1000); </pre>	
参数	[in] lLoginID	登录句柄
	[in] emType	控制类型
	[in] pInParam	输入参数, 根据 emType 的不同而选择不同的结构体
	[out] pOutParam	返回参数, 根据 emType 的不同而选择不同的结构体
	[in] waittime	超时时间
返回值	成功返回 TRUE, 失败返回 FALSE	
说明	无	

10.1 断线回调函数 fDisConnect

选项	说明	
描述	断线回调函数	
函数	typedef void (CALLBACK *fDisConnect)(LLONG ILoginID, char* pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
参数	[out] ILoginID	CLIENT_LoginEx2 的返回值
	[out] pchDVRIP	断线的设备 IP
	[out] nDVRPort	断线的设备端口
	[out] dwUser	回调函数的用户参数
返回值	无	
说明	无	

10.2 断线重连回调函数 fHaveReConnect

选项	说明	
描述	断线重连回调函数	
函数	typedef void (CALLBACK *fHaveReConnect)(LLONG ILoginID, char* pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
参数	[out] ILoginID	CLIENT_LoginEx2 的返回值
	[out] pchDVRIP	断线后重连成功的设备 IP
	[out] nDVRPort	断线后重连成功的设备端口
	[out] dwUser	回调函数的用户参数
返回值	无	
说明	无	

10.3 实时监视数据回调函数 fRealDataCallBackEx

选项	说明	
描述	实时监视数据回调函数	
函数	<pre>typedef void (CALLBACK *fRealDataCallBackEx)(LONG lRealHandle, DWORD dwDataType, BYTE* pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser);</pre>	
参数	[out] lRealHandle	CLIENT_RealPlayEx 的返回值
	[out] dwDataType	数据类型，0 表示原始数据，2 表示 YUV 数据
	[out] pBuffer	监视数据块地址
	[out] dwBufSize	监视数据块的长度，单位：字节
	[out] param	回调数据参数结构体，dwDataType 值不同类型不同 dwDataType 为 0 时，param 为空指针 dwDataType 为 2 时，param 为 tagCBYUVDataParam 结构体指针
	[out] dwUser	回调函数的用户参数
返回值	无	
说明	无	

10.4 智能事件回调函数 fAnalyzerDataCallBack

选项	说明	
描述	智能事件回调函数	
函数	<pre>typedef int (CALLBACK *fAnalyzerDataCallBack)(LONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE* pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void* reserved);</pre>	
参数	[out] lAnalyzerHandle	CLIENT_RealLoadPictureEx 返回值
	[out] dwAlarmType	智能事件类型
	[out] pAlarmInfo	事件信息缓存
	[out] pBuffer	图片缓存
	[out] dwBufSize	图片缓存大小

选项	说明	
	[out] dwUser	用户数据
	[out] nSequence	nSequence 表示上传的相同图片情况，值为 0 时表示第一次出现，值为 2 表示最后一次出现或仅出现一次，值为 1 表示此次之后还有
	[out] reserved	保留
返回值	无	
说明	无	

10.5 回放及按文件下载进度回调函数 fDownloadPosCallBack

选项	说明	
描述	回放及按文件下载进度回调函数	
函数	<pre>typedef void (CALLBACK *fDownloadPosCallBack)(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownloadSize, LDWORD dwUser);</pre>	
参数	[out]lPlayHandle	回放或下载接口返回值
	[out]dwTotalSize	总大小，单位：KB
	[out]dwDownloadSize	已下载的大小，单位：KB <ul style="list-style-type: none"> • -1：本次回放结束 • -2：写文件失败
	[out]dwUser	用户数据
返回值	无	
说明	无	

10.6 回放及下载数据回调函数 fDataCallBack

选项	说明	
描述	回放及下载数据回调函数	
函数	<pre>typedef int (CALLBACK *fDataCallBack)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);</pre>	
参数	[out]lPlayHandle	回放或下载接口返回值
	[out] dwDataType	这里为 0（原始数据）
	[out] pBuffer	数据缓冲

选项	说明	
	[out] dwBufSize	缓冲长度，单位：字节
	[out] dwUser	用户数据
返回值	无	
说明	无	

10.7 人脸查询进度回调函数 fFaceFindState

选项	说明	
描述	人脸查询进度回调函数	
函数	<pre>typedef void (CALLBACK *fFaceFindState)(LONG lLoginID, LONG lAttachHandle, NET_CB_FACE_FIND_STATE* pstStates, int nStateNum, LDWORD dwUser);</pre>	
参数	[out] lLoginID	返回登录句柄
	[out] lAttachHandle	事件订阅句柄
	[out] pstStates	人脸查询的状态信息
	[out] nStateNum	人脸查询的进度
	[out] dwUser	用户数据
返回值	无	
说明	无	

10.8 客流量事件订阅回调 fVideoStatSumCallBack

选项	说明	
描述	客流量事件订阅回调	
函数	<pre>typedef void (CALLBACK *fVideoStatSumCallBack) (LONG lAttachHandle, NET_VIDEOSTAT_SUMMARY* pBuf, DWORD dwBufLen, LDWORD dwUser);</pre>	
参数	[out] lAttachHandle	客流订阅句柄
	[out] pBuf	客流返回数据
	[out] dwBufLen	返回数据长度
	[out] dwUser	用户数据
返回值	无	
说明	无	

10.9 批量下载文件进度回调函数 fMultiFileDownloadPosCB

选项	说明	
描述	批量下载文件进度回调函数	
函数	<pre>typedef void (CALLBACK *fMultiFileDownloadPosCB)(LLONG IDownloadHandle, DWORD dwID, DWORD dwFileTotalSize, DWORD dwDownloadSize, int nError, LDWORD dwUser, void* pReserved);</pre>	
参数	[out] IDownloadHandle	批量下载文件句柄
	[out] dwID	ID 为用户设置的 dwFileID
	[out] dwFileTotalSize	下载文件的总大小
	[out] dwDownloadSize	当前文件下载大小，当该值为该类型的最大值时表示下载结束
	[out] nError	下载出错：1-缓存不足，2-对返回数据的校验出错，3-下载当前文件失败，4-创建对应保存文件失败
	[out] dwUser	用户数据
	[out] pReserved	保留字段
返回值	无	
说明	无	