

NetSDK_JAVA

编程指导手册



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。

本文档描述了网络摄像机（IP Camera，简称 IPC）、球机（Speed Dome，简称 SD）和热成像（Thermal Camera，简称 TPC）产品的通用业务涉及的 SDK 接口以及调用流程，更多功能接口、结构体等请参考《网络 SDK 开发手册》。











本文档提供的示例代码仅为演示接口调用方法，不保证能直接拷贝编译。

读者对象

使用 SDK 的软件开发工程师、产品经理和项目经理等。

符号约定

在本文档中可能出现下列标识，代表的含义如下。

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

修订记录

版本号	修订内容	发布日期
V2.0.1	新增远程通道模块	2025.07
V2.0.0	登录模块优化	2025.02
V1.0.4	新增转码相关章节，新增云台控制 Param4 枚举。	2024.07
V1.0.3	新增动态库路径下的系统权限。	2023.07
V1.0.2	全文优化语言。	2023.02
V1.0.1	更新依赖库信息。	2021.04
V1.0.0	首次发布。	2020.03

名词解释

以下对本文档中使用的专业名词分别说明，帮助您更好的理解各个业务功能。

名词	说明
主码流	视频码流类型的一种，一般是分辨率比较高，清晰度、画质更好的码流，在网络资源不受限的前提下能得到更好的体验。
辅码流	较主码流分辨率、清晰度都低一些，但占用的网络资源少；用户可以根据不同的适用场景选择不同的码流类型。
视频通道	设备的每路视频从 0 开始编号，得到的号码叫做视频通道号（简称通道），目前前端产品除热成像设备（TPC）外，一般只有一路通道，所以通道号一般为 0。
登录句柄	访问设备第一步需要登录（即鉴权），登录成功后获取到的唯一的 ID 号，标识此次登录会话的句柄，后续其他业务均需要用到该句柄，句柄直到用户登出（注销）失效。
相对定位	云台控制中快速定位的一种方法，通过提供云台坐标（X,Y 轴坐标）的差值给设备，设备根据当前云台位置和用户指定的差值，计算并转到最终位置；同时支持 ZOOM 控制。
绝对定位	云台控制中快速定位的一种方法，通过提供确定的水平、垂直坐标（角度坐标）给设备，设备直接转到用户指定的坐标位置，同时支持 ZOOM 控制。
PCM	脉冲编码调制（Pulse Code Modulation），是数字通信的编码方式之一，将模拟信号转换成数字信号的无损编码方式，适用于对数据传输率要求较高、需要更高宽带的用户使用。
PTZ	Pan Tilt Zoom，指云台全方位移动及镜头变倍、变焦控制。

目录

前言.....	I
名词解释.....	III
第 1 章 内容简介	1
1.1 概述	1
1.2 适用性.....	1
1.3 库加载问题及解决方法	2
1.3.1 Windows 环境下使用 Java 项目	2
1.3.2 Linux 环境下使用 Java 项目	3
1.3.3 将项目作为 jar 包执行	5
1.3.4 动态库路径下的系统权限	6
1.4 旧版本 jna 升级至新版本	6
第 2 章 主要功能	8
2.1 SDK 初始化	8
2.1.1 简介	8
2.1.2 接口总览.....	8
2.1.3 流程说明.....	8
2.1.4 示例代码.....	10
2.2 设备登录	12
2.2.1 简介	12
2.2.2 接口总览.....	12
2.2.3 流程说明.....	12
2.2.4 示例代码.....	14
2.3 实时预览	17
2.3.1 简介	17
2.3.2 接口总览.....	17
2.3.3 流程说明.....	18
2.3.4 示例代码.....	22
2.4 抓图	24
2.4.1 简介	24
2.4.2 接口总览.....	24
2.4.3 流程说明.....	24
2.4.4 示例代码.....	27
2.5 云台控制	30
2.5.1 简介	30
2.5.2 接口总览.....	30
2.5.3 流程说明.....	30
2.5.4 示例代码.....	32
2.6 语音对讲.....	33
2.6.1 简介	33
2.6.2 接口总览.....	33
2.6.3 流程说明.....	33
2.6.4 示例代码.....	35
2.7 报警监听	38

2.7.1 简介	38
2.7.2 接口总览	38
2.7.3 流程说明	38
2.7.4 示例代码	40
2.8 智能订阅	41
2.8.1 简介	41
2.8.2 接口总览	41
2.8.3 流程说明	41
2.8.4 示例代码	43
2.9 录像回放	46
2.9.1 简介	46
2.9.2 接口总览	46
2.9.3 流程说明	47
2.9.4 示例代码	48
2.10 录像下载	51
2.10.1 简介	51
2.10.2 接口总览	51
2.10.3 流程说明	52
2.10.4 示例代码	53
2.11 实时预览转码	56
2.11.1 简介	56
2.11.2 接口总览	56
2.11.3 流程说明	57
2.11.4 示例代码	57
2.12 录像回放转码	60
2.12.1 简介	60
2.12.2 接口总览	60
2.12.3 流程说明	60
2.12.4 示例代码	61
2.13 录像下载转码	65
2.13.1 简介	65
2.13.2 接口总览	65
2.13.3 流程说明	65
2.13.4 示例代码	66
2.14 远程通道	69
2.14.1 简介	69
2.14.2 接口总览	69
2.14.3 流程说明	70
2.14.4 示例代码	70
第 3 章 接口函数	74
3.1 SDK 初始化	74
3.1.1 SDK 初始化 CLIENT_Init	74
3.1.2 SDK 清理 CLIENT_Cleanup	74
3.1.3 设置断线重连回调函数 CLIENT_SetAutoReconnect	74
3.1.4 设置网络参数 CLIENT_SetNetworkParam	75
3.2 设备登录	75
3.2.1 用户登录设备 CLIENT_LoginWithHighLevelSecurity	75

3.2.2 用户登出设备 CLIENT_Logout	75
3.3 实时预览	76
3.3.1 打开预览 CLIENT_RealPlayEx	76
3.3.2 关闭预览 CLIENT_StopRealPlayEx	76
3.3.3 保存预览数据 CLIENT_SaveRealData	77
3.3.4 停止保存预览数据 CLIENT_StopSaveRealData	77
3.3.5 设置预览数据回调 CLIENT_SetRealDataCallBackEx	77
3.4 抓图	78
3.4.1 同步抓图 CLIENT_SnapPictureToFile	78
3.4.2 本地抓图 CLIENT_CapturePictureEx	78
3.4.3 异步抓图 CLIENT_SnapPictureEx	79
3.4.4 设置异步抓图回调	79
3.5 云台控制	79
3.5.1 云台控制 CLIENT_DHPTZControlEx2	79
3.6 语音对讲	83
3.6.1 开启对讲 CLIENT_StartTalkEx	83
3.6.2 关闭对讲 CLIENT_StopTalkEx	83
3.6.3 发送语音 CLIENT_TalkSendData	84
3.6.4 解码语音 CLIENT_AudioDecEx	84
3.7 报警监听	84
3.7.1 开始报警监听 CLIENT_StartListenEx	84
3.7.2 停止报警监听 CLIENT_StopListen	85
3.7.3 设置报警回调 CLIENT_SetDVRMessCallBack	85
3.8 智能订阅	85
3.8.1 开始智能事件订阅 CLIENT_RealLoadPictureEx	85
3.8.2 停止智能事件订阅 CLIENT_StopLoadPic	86
3.9 录像回放	87
3.9.1 按时间方式回放 CLIENT_PlayBackByTimeEx	87
3.9.2 设置工作模式 CLIENT_SetDeviceMode	87
3.9.3 停止录像回放 CLIENT_StopPlayBack	88
3.9.4 暂停或恢复录像回放 CLIENT_PausePlayBack	88
3.10 录像下载	88
3.10.1 查询时间段内的所有录像文件 CLIENT_QueryRecordFile	88
3.10.2 按时间下载录像 CLIENT_DownloadByTimeEx	90
3.10.3 停止录像下载 CLIENT_StopDownload	90
3.11 开启实时预览转码接口 CLIENT_RealPlayByDataType	92
3.12 开启录像回放转码接口 CLIENT_PlayBackByDataType	92
3.13 开启录像下载转码接口 CLIENT_DownloadByDataType	92
3.14 远程通道	93
3.14.1 获取远程设备信息 CLIENT_MatrixGetCameras	93
3.14.2 配置远程设备信息 CLIENT_MatrixSetCameras	93
第 4 章 回调函数	94
4.1 断线回调函数 fDisConnect	94
4.2 断线重连回调函数 fHaveReConnect	94
4.3 实时预览数据回调函数 fRealDataCallBackEx	95
4.4 音频数据回调函数 pfAudioDataCallBack	95
4.5 智能事件回调 fAnalyzerDataCallBack	96

4.6 按时间下载回调函数 fTimeDownLoadPosCallBack..... 96

4.7 报警订阅回调函数 fMessCallBack 97

4.8 异步抓图回调 fSnapRev 98

4.9 实时预览转码数据回调函数 fDataCallBackEx..... 99

4.10 回放进度回调函数 fDownLoadPosCallBack 100

4.11 回放数据回调 fDataCallBack 100

附录 1 法律声明 101

附录 2 网络安全建议..... 102

第 1 章 内容简介

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：SDK 初始化、设备登录、实时预览、云台控制、语音对讲、报警监听、智能订阅、录像回放和录像下载等。

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如下：

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.dll	库文件
	avnetsdk.dll	库文件
	dhconfigsdk.h	头文件
	dhconfigsdk.dll	库文件
播放（编码解码）辅助库	dhplay.dll	播放库
	StreamConvertor.dll	转码库

- Linux 开发包所包含的文件如下：

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
	dhconfigsdk.h	头文件
	libdhconfigsdk.so	库文件

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。

1.2 适用性

- 推荐内存：不低于 512 MB。
- Jdk 使用版本：jdk1.6；jdk1.8
- SDK 支持的系统：
 - Windows
Windows 10/Windows 8.1/Windows 7/vista/XP/2000 以及 Windows Server 2008/2003。

- ◇ Linux
Red Hat/SUSE 等通用 Linux 系统。

1.3 库加载问题及解决方法

目前提供 Windows(.dll)、Linux(.so)两种平台的动态库。其中 win 和 linux 分为 64 位、32 位版本。而调用 C++动态库的主要的方式分为“直接使用 Java 项目”和“将 Java 项目作为其他项目的 jar 包依赖运行”两种。在加载库的过程中会出现“找不到动态库”的相关错误。

“找不到动态库”问题的根本原因为代码路径和物理路径不匹配。因为 linux 版本的动态库名称相较于 Windows 版本多出 lib 前缀，故 Linux 环境下加载动态库需要注意 lib 的前缀，在拼接动态库路径时需要加上 lib 前缀。使用 java.io.tmpdir 方式实现路径映射时，需要注意此种方式优先级较低。

1.3.1 Windows 环境下使用 Java 项目

可能的错误信息：

```
[Load dhnetsdk path : ./wrongpath/libs/win64/dhnetsdk]
Exception in thread "AWT-EventQueue-0"java.lang.UnsatisfiedLinkError: Unable to load
library './wrongpath/libs/win64/dhnetsdk': 找不到指定的模块。
at com.sun.jna.NativeLibrary.loadLibrary(NativeLibrary.java:169)
at com.sun.jna.NativeLibrary.getInstance(NativeLibrary.java:242)
at com.sun.jna.Library$Handler.<init>(Library.java:140)
```

出现上面的报错信息时，可以按下面代码定位错误的位置。

```
public interface NetSDKLib extends Library {
    NetSDKLib NETSDK_INSTANCE =
Native.load(LibraryLoad.getLoadLibrary("dhnetsdk"), NetSDKLib.class);
~~~~~ LibraryLoad. getLibraryFold 代码如下
    // 获取系统对应的动态库文件夹
    private static String getLibraryFold() {
        String osType;
        String osName = System.getProperty("os.name");
        if (osName.toLowerCase().startsWith("linux")) {
            osType = ARCH_LINUX;
        } else if (osName.toLowerCase().startsWith("mac"))
            || osName.toLowerCase().startsWith("darwin")) {
            osType = ARCH_MAC;
        } else if (osName.toLowerCase().startsWith("windows")) {
            osType = ARCH_WINDOWS;
        } else {
            osType = "";
        }
    }
```

```

String arch = System.getProperty("os.arch");
arch = arch.toLowerCase().trim();
if ("i386".equals(arch) || "i686".equals(arch) || "x86".equals(arch)) {
    arch = PREFIX_32 + "";
} else if ("x86_64".equals(arch) || "amd64".equals(arch)) {
    arch = PREFIX_64 + "";
} else if (arch.startsWith("arm")) {
    arch = PREFIX_ARM + "";
} else {
    arch = PREFIX_ARM + "";
}
System.out.println("动态库文件夹:"+osType + arch);
return osType + arch;
}

```

解决途径

出现错误的原因一般是 jdk 版本与系统环境不匹配：如果 java 项目的 jdk 是 32 位，系统环境是 64 位，而 sdk 是 64 位，将会出现上述问题。sdk 项目的版本、jdk 的版本、操作系统的版本三者需要保持一致，同为 win64 或者同为 win32。

说明

Windows 平台出错的可能性较低，此处只是举例。请注意平台必须和动态库对应，Windows 对应的动态库后缀为.dll。

1.3.2 Linux 环境下使用 Java 项目

Linux 环境下通过脚本代码运行示例（脚本代码路径为根目录下的 run.sh）

```

[user@localhost ~]# ./run.sh com/netsdk/demo/customize/ConfigDemo
--> ClassPath: ../resources/jna.jar:../resources/gson-2.6.2.jar:../resources/fastjson-1.2.70.jar:../resources/INetSDK.jar:../resources/dynamic-lib-load.xml:
--> Bin ../bin
Create ../bin.
--> linux 64 System.
cp: 无法获取'../resources/linux64' 的文件状态(stat): 没有那个文件或目录
--> path: pwd
load library: /tmp/libdhnetsdk.so
Exception in thread "main" java.lang.UnsatisfiedLinkError: Unable to load library
'/tmp/libdhnetsdk.so':
/tmp/libdhnetsdk.so: 无法打开共享对象文件: 没有那个文件或目录
/tmp/libdhnetsdk.so: 无法打开共享对象文件: 没有那个文件或目录

```

```

Native library (tmp/libdhnetsdk.so) not found in resource path
(..resources/jna.jar:../resources/gson-2.6.2.jar:../resources/fastjson-1.2.70.jar:../resources/INetSDK.jar:../resources/dynamic-lib-load.xml::)
    at com.sun.jna.NativeLibrary.loadLibrary(NativeLibrary.java:302)
    at com.sun.jna.NativeLibrary.getInstance(NativeLibrary.java:455)
    at com.sun.jna.Library$Handler.<init>(Library.java:192)
    at com.sun.jna.Native.load(Native.java:596)
    at com.sun.jna.Native.load(Native.java:570)
    at com.netsdk.lib.NetSDKLib.<clinit>(NetSDKLib.java:20)
    at com.netsdk.demo.customize.ConfigDemo.<init>(ConfigDemo.java:30)
    at com.netsdk.demo.customize.ConfigDemo.main(ConfigDemo.java:48)
    Suppressed: java.lang.UnsatisfiedLinkError: /tmp/libdhnetsdk.so: 无法打开共享对象
    文件: 没有那个文件或目录
        at com.sun.jna.Native.open(Native Method)
        at com.sun.jna.NativeLibrary.loadLibrary(NativeLibrary.java:191)
        ... 7 more
    Suppressed: java.lang.UnsatisfiedLinkError: /tmp/libdhnetsdk.so: 无法打开共享对象
    文件: 没有那个文件或目录
        at com.sun.jna.Native.open(Native Method)
        at com.sun.jna.NativeLibrary.loadLibrary(NativeLibrary.java:204)
        ... 7 more
    Suppressed: java.io.IOException: Native library (tmp/libdhnetsdk.so) not found in
    resource path (../resources/jna.jar:../resources/gson-2.6.2.jar:../res
    ources/fastjson-1.2.70.jar:../resources/INetSDK.jar:../resources/dynamic-lib-loa
    d.xml::)
        at com.sun.jna.Native.extractFromResourcePath(Native.java:1095)
        at com.sun.jna.NativeLibrary.loadLibrary(NativeLibrary.java:276)
        ... 7 more

```

run.sh 脚本部分代码

```

##指定库的路径    加入动态链接库
if [[ $os =~ "Darwin" ]]; then
    #CP+=../libs/mac64
    echo "--> mac 64 System"
    cp -r ../resources/mac64 $BIN/mac64
elif [ $(getconf LONG_BIT) = '64' ]; then
    echo "--> linux 64 System."
    #export LD_LIBRARY_PATH=../resources/linux64
    cp -r ../resources/linux64 $BIN/linux64

```

```

else
    echo "--> linux 32 System."
    #export LD_LIBRARY_PATH=./libs/linux32
    cp -r ../resources/linux32 $BIN/linux32
fi

```

注意事项

此处同 windows 类似。不过如果报错找不到动态库，在确定动态库无误的情况下，请检查脚本中引入的 PATH 路径是否和系统路径一致。

解决方式

使用如下任意一个方式解决。

- 方式一：同 windows 的方法，检查环境配置是否一致。
- 方式二：临时文件夹加载方式—java.io.tmpdir，此方法适用于多平台。调用 public static void setExtractPath(String path) 方法中的 path 参数改为 绝对路径。
 1. 拷贝所需动态库到某文件夹，假设 D:/win64/
 2. 使用 static 语句块调用方法：（路径前后一致）

```

static{
    LibraryLoad.setExtractPath(String path) };
}

```

1.3.3 将项目作为 jar 包执行

将动态库内置于 jar 包的好处是：只要导出 jar 包，无论平台之间有什么差异，直接执行 jar 即可。

此处给出一种内置 jar 包的方式：将 jar 包中动态库写入本机临时文件夹（java.io.tmpdir），再从本机读取动态库载入内存，以下为测试代码。

```

public interface NetSDKLib extends Library {
    NetSDKLib NETSDK_INSTANCE = Native.load(LibraryLoad.getLoadLibrary("dhnetsdk"),
        NetSDKLib.class);
    NetSDKLib CONFIG_INSTANCE = Native.load(LibraryLoad.getLoadLibrary("dhconfigsdk"),
        NetSDKLib.class);

    public class LibraryLoad {
        public static String getLoadLibrary(String libraryName) {
            currentFold = getLibraryFold();
            if (dynamicParseUtil == null) {
                try {
                    dynamicParseUtil =
                        new DynamicParseUtil(
                            LibraryLoad.class.getClassLoader().getResourceAsStream("dynamic-lib-load.xml"));
                    if (!written) {

```

```

        for (String libName : dynamicParseUtil.getLibsSystem(currentFold)) {
            extractLibrary(libName);
        }
        written = true;
    }
} catch (ParserConfigurationException | IOException | SAXException e) {
    e.printStackTrace();
}
}
String fullName = getLibraryName(libraryName);
String path = EXTRACT_PATH;
if (!(EXTRACT_PATH.endsWith("/") || EXTRACT_PATH.endsWith("\\"))) {
    path = EXTRACT_PATH + "/";
}
System.out.println("load library: " + path + fullName);
return path + fullName;
}
}

```

1.3.4 动态库路径下的系统权限

在 windows 和 linux 中，Java 项目会将动态库复制到系统的临时目录中，临时目录路径可以通过 LibraryLoad 类中如下的代码函数获取。



说明

需要保证该路径下动态库文件有读写权限。若上述临时目录下的动态库文件没有赋读写权限，会导致加载动态库失败。

```

String fullName = getLibraryName(libraryName);
String path = EXTRACT_PATH;
if (!(EXTRACT_PATH.endsWith("/") || EXTRACT_PATH.endsWith("\\"))) {
    path = EXTRACT_PATH + "/";
}
System.out.println("load library: " + path + fullName);

```

1.4 旧版本 jna 升级至新版本

步骤1 将 resources/路径下旧版本的 jna 包替换成新版本的 jna 包。将新版本的 jna 配置到当前环境中。

步骤2 将 NetSDKLib 封装类中加载库的方法改成如下：

```

NetSDKLib NETSDK_INSTANCE = Native.load(LibraryLoad.getLoadLibrary("dhnetsdk"),
NetSDKLib.class);

```

步骤3 将 NativeString 类中 Pointer.getString 方法和 Pointer.setString 方法中的参数改为 pointer.getString(0)和 pointer.setString(0, string)。



注意

jna 版本升级之后，如果运行的 demo 中存在回调函数并且继承了 StdCallCallback，需将 StdCallCallback 注释掉，否则在 linux 环境下运行，会出现崩溃问题。

第 2 章 主要功能

按实现功能的不同可以分成 10 个模块，实现每个模块的功能均需要初始化 SDK、设备登录、注销用户和释放 SDK 资源 4 个步骤。可选流程不会影响其他流程的功能使用。

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

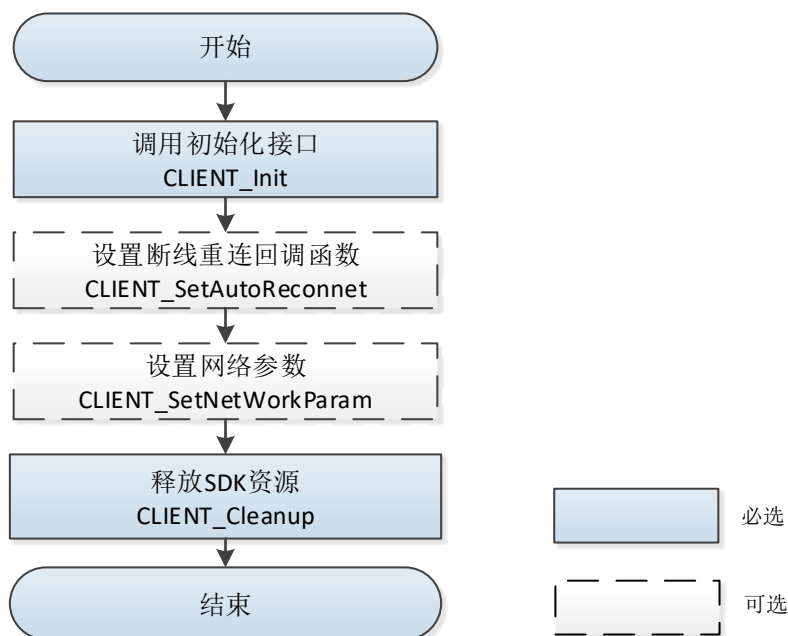
表2-1 SDK 初始化接口信息

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 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，支持多线程多次成对调用，但建议全局调用一次。
- 初始化：`CLIENT_Init` 接口内部多次调用时，仅在内部用做计数，不会重复申请资源。
- 清理：`CLIENT_Cleanup` 接口内会清理所有已开启的业务，如登录、实时预览和报警订阅等。
- 断线重连：SDK 可以设置断线重连功能，当遇到一些特殊情况（例如断网、断电等）设备断线时，在 SDK 内部会定时持续不断地进行登录操作，直至成功登录设备。断线重连后可以恢复实时预览和录像回放业务，其他业务无法恢复。
- 加载动态库：如果加载动态库遇到报错如“Unable to load library 'C:/wrongpath/libs/win64/dhnetSDK': 找不到指定的模块”，通常是路径不匹配，需要根据报错信息调整动态库的位置或者修改代码。此问题多见于打包整个工程为 jar 包提供给其他项目时。由于此问题跟平台和工程使用方式相关，不能一概而论，需要具体分析。比如在 linux 平台下直接使用工程可以通过以下方式将动态库路径加载到动态库搜索路径中。
 - 在终端输入：`export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/XXX` 当前终端生效
 - 修改 `~/.bashrc` 或 `~/.bash_profile`，最后一行添加 `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/XXX`，保存之后，使用 `source .bashrc` 执行该文件，当前用户生效。
 - 修改 `/etc/profile`，添加内容如第 2 条，同样保存后用 `source` 执行该文件，所有用户生效。

效。

2.1.4 示例代码

```
import java.io.File;

import main.java.com.netsdk.lib.NetSDKLib;
import main.java.com.netsdk.lib.NetSDKLib.LLong;
import main.java.com.netsdk.lib.ToolKits;

import com.sun.jna.ptr.IntByReference;

/**
 * 登录接口实现
 * 主要有：初始化、登录、登出功能
 */
public class LoginModule {

    public static NetSDKLib netsdk          = NetSDKLib.NETSDK_INSTANCE;
    public static NetSDKLib configsdk       = NetSDKLib.CONFIG_INSTANCE;

    // 登录句柄
    public static LLong m_hLoginHandle = new LLong(0);

    private static boolean blnit          = false;
    private static boolean bLogopen = false;

    //初始化
    public static boolean init(NetSDKLib.fDisconnect disconnect,
NetSDKLib.fHaveReConnect haveReConnect) {
        blnit = netsdk.CLIENT_Init(disconnect, null);
        if(!blnit) {
            System.out.println("Initialize SDK failed");
            return false;
        }

        //打开日志，可选
        NetSDKLib.LOG_SET_PRINT_INFO setLog = new
NetSDKLib.LOG_SET_PRINT_INFO();
        File path = new File("./sdklog/");
```

```

        if (!path.exists()) {
            path.mkdir();
        }
        String logPath = path.getAbsolutePath().getParent() + "\\sdklog\\" + ToolKits.getDate()
+ ".log";
        setLog.nPrintStrategy = 0;
        setLog.bSetFilePath = 1;
        System.arraycopy(logPath.getBytes(), 0, setLog.szLogFilePath, 0,
logPath.getBytes().length);
        System.out.println(logPath);
        setLog.bSetPrintStrategy = 1;
        bLogopen = netsdk.CLIENT_LogOpen(setLog);
        if(!bLogopen ) {
            System.err.println("Failed to open NetSDK log");
        }

        // 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
        netsdk.CLIENT_SetAutoReconnect(haveReConnect, null);

        //设置登录超时时间和尝试次数，可选
        int waitTime = 5000; //登录请求响应超时时间设置为 5S
        int tryTimes = 1;    //登录时尝试建立链接 1 次
        netsdk.CLIENT_SetConnectTime(waitTime, tryTimes);

        // 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,可选
        NetSDKLib.NET_PARAM netParam = new NetSDKLib.NET_PARAM();
        netParam.nConnectTime = 10000;    // 登录时尝试建立链接的超时时间
        netParam.nGetConnInfoTime = 3000; // 设置子连接的超时时间
        netsdk.CLIENT_SetNetworkParam(netParam);

        return true;
    }

    //清除环境
    public static void cleanup() {
        if(bLogopen) {

```

```
        netsdk.CLIENT_LogClose();
    }

    if(blNit) {
        netsdk.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

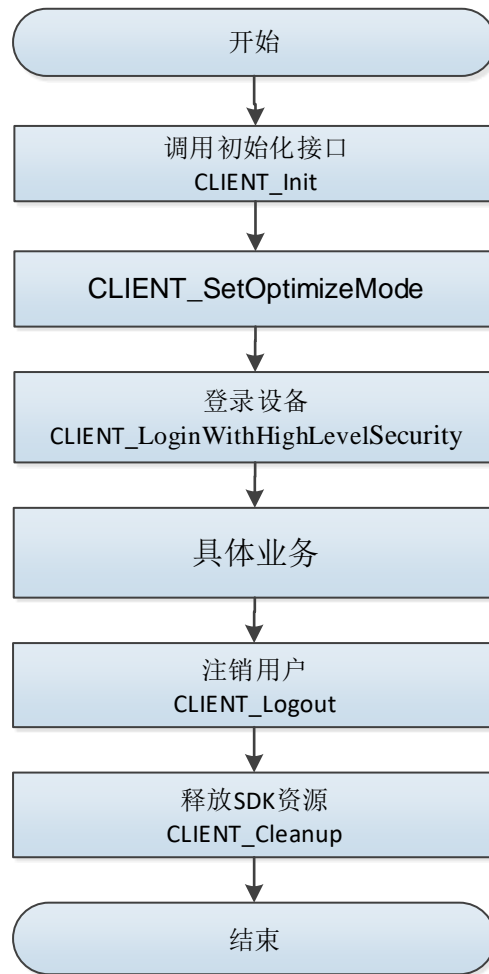
表2-2 设备登录接口信息

接口	说明
CLIENT_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_Logout	登出接口
CLIENT_SetOptimizeMode	优化获取硬盘信息

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 可选操作，优化获取硬盘信息，调用 `CLIENT_SetOptimizeMode` 接口。
- 步骤3 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` 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

- 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。例如打开预览后，在不需要使用预览时，用户应该调用结束预览的接口。
- 登录与登出配对使用，登录会消耗一定的内存和 socket 信息，在登出后释放资源。
- 登录失败：建议通过登录接口的 `error` 参数（登录错误码）初步排查。常见错误码请参见表

- 2-3。
- 多设备登录: SDK 初始化后, 可以登录多台设备, 但是相应的登录句柄、登录信息需要调整。

表2-3 常见错误码及含义

error 的错误码	含义
1	密码不正确
2	用户名不存在
3	登录超时。规避示例代码如下: NET_PARAM stuNetParam = {0}; stuNetParam.nWaittime = 8000; // unit ms CLIENT_SetNetworkParam (&stuNetParam);
4	账号已登录
5	账号已被锁定
6	账号被列为禁止名单
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

2.2.4 示例代码

```
import java.io.File;

import main.java.com.netsdk.lib.NetSDKLib;
import main.java.com.netsdk.lib.NetSDKLib.LLong;
import main.java.com.netsdk.lib.ToolKits;

import com.sun.jna.ptr.IntByReference;

public class LoginModule {

    public static NetSDKLib netsdk          = NetSDKLib.NETSDK_INSTANCE;
    public static NetSDKLib configsdk      = NetSDKLib.CONFIG_INSTANCE;

    //SDK 初始化, SDK 清理省略

    public static void InitTest(){
        // 初始化 SDK 库
        netSdk.CLIENT_Init(DisconnectCallback.getInstance(), null);
    }
}
```

```

        // 设置断线重连成功回调函数
        netSdk.CLIENT_SetAutoReconnect(HaveReconnectCallback.getInstance(), null);
        // 减少登录时的信息查询以提升速度
        int mode =
EM_OPTTYPE_MOBILE_TYPE.OPTTYPE_MOBILE_DEVICE_ATTR.getValue() |
EM_OPTTYPE_MOBILE_TYPE.OPTTYPE_MOBILE_DEVICE_SN.getValue()
        |
EM_OPTTYPE_MOBILE_TYPE.OPTTYPE_MOBILE_DISK_INFO.getValue();
        // 创建 IntByReference 对象，将 mode 赋值给它
        IntByReference modeReference = new IntByReference(mode);

netSdk.CLIENT_SetOptimizeMode(EM_OPTIMIZE_TYPE.EM_OPT_TYPE_MOBILE_OPTIO
N.getValue(), modeReference.getPointer());
        //打开日志，可选 0
        NetSDKLib.LOG_SET_PRINT_INFO setLog = new
NetSDKLib.LOG_SET_PRINT_INFO();
        String logPath = new File(".").getAbsolutePath().getParent() + File.separator +
"sdk_log" + File.separator + "sdk.log";
        setLog.bSetFilePath = 1;
        System.arraycopy(logPath.getBytes(), 0, setLog.szLogFilePath, 0,
logPath.getBytes().length);
        setLog.bSetPrintStrategy = 1;
        setLog.nPrintStrategy = 0;
        if (!netSdk.CLIENT_LogOpen(setLog)){
            System.err.println("Open SDK Log Failed!!!");
        }
    }
    /**
     * 设备断线回调
     */
    private static class DisconnectCallback implements NetSDKLib.fDisConnect {
        private static DisconnectCallback instance = new DisconnectCallback();

        private DisconnectCallback() {
        }

        public static DisconnectCallback getInstance() {
            return instance;
        }

        public void invoke(NetSDKLib.LLong lLoginID, String pchDVRIP, int nDVRPort,
Pointer dwUser) {
            System.out.printf("Device[%s:%d] Disconnect!\n", pchDVRIP, nDVRPort);
        }
    }
    /**
     * 设备重连回调

```

```

*/
private static class HaveReconnectCallback implements NetSDKLib.fHaveReConnect {
    private static HaveReconnectCallback instance = new HaveReconnectCallback();

    private HaveReconnectCallback() {
    }

    public static HaveReconnectCallback getInstance() {
        return instance;
    }

    public void invoke(NetSDKLib.LLong lLoginID, String pchDVRIP, int nDVRPort,
Pointer dwUser) {
        System.out.printf("Device[%s:%d] HaveReconnected!\n", pchDVRIP, nDVRPort);
    }
}

// 设备信息
public static NetSDKLib.NET_DEVICEINFO_Ex m_stDeviceInfo = new
NetSDKLib.NET_DEVICEINFO_Ex();

//登录句柄
public static LLong m_hLoginHandle = new LLong(0);

//登录设备
public static boolean login(String m_strIp, int m_nPort, String m_strUser, String
m_strPassword) {
    //入参
    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY pstInParam=
new NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY();
    pstInParam.szIP= m_strIp;
    pstInParam.nport= m_nPort;
    pstInParam.szUserName= m_strUser;
    pstInParam.szPassword= m_strPassword;
    //出参
    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY pstOutParam=
new NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY();
    m_hLoginHandle =
netsdk.CLIENT_LoginWithHighLevelSecurity(NET_IN_LOGIN_WITH_HIGHLEVEL_SECUKIT
Y pstInParam, NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY pstOutParam);

    if(m_hLoginHandle.longValue() == 0) {
        System.err.printf("Login Device[%s] Port[%d]Failed. %s\n", m_strIp, m_nPort,

```



```

ToolKits.getErrorCodePrint());
    } else {
        System.out.println("Login Success ");
    }

    return m_hLoginHandle.longValue() == 0? false:true;
}

//登出设备
public static boolean logout() {
    if(m_hLoginHandle.longValue() == 0) {
        return false;
    }

    boolean bRet = netsdk.CLIENT_Logout(m_hLoginHandle);
    if(bRet) {
        m_hLoginHandle.setValue(0);
    }

    return bRet;
}
}

```

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 实时预览接口信息

接口	说明
CLIENT_RealPlayEx	开始实时预览扩展接口
CLIENT_StopRealPlayEx	停止实时预览扩展接口
CLIENT_SaveRealData	开始本地保存实时预览数据

接口	说明
CLIENT_StopSaveRealData	停止本地保存实时预览数据
CLIENT_SetRealDataCallBackEx	设置实时预览数据回调函数扩展接口

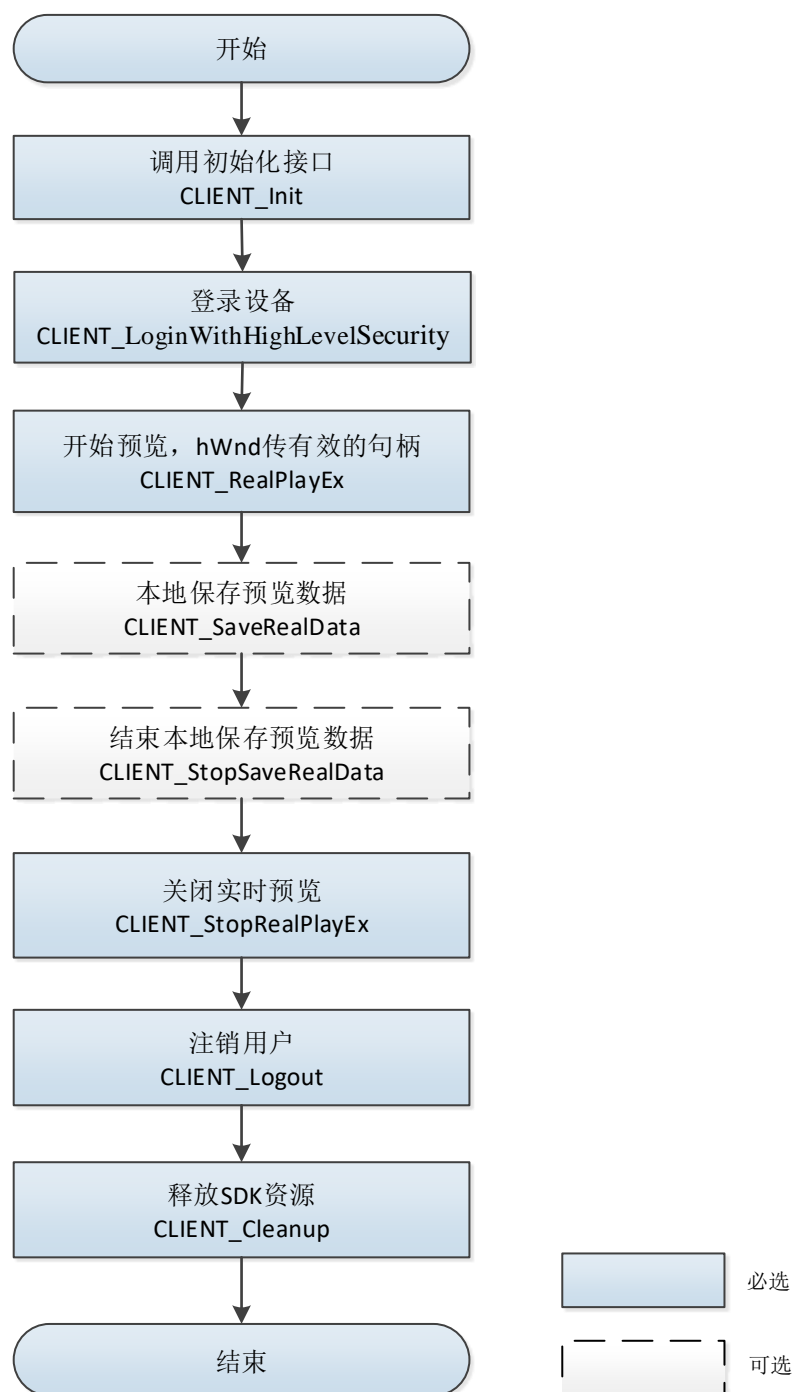
2.3.3 流程说明

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 解码播放

SDK 内部调用辅助库里的 **PlaySDK** 库实现实时播放。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_LoginWithHighLevelSecurity** 登录设备。
- 步骤3 调用 **CLIENT_RealPlayEx** 启动实时预览，参数 **hWnd** 为有效窗口句柄。
- 步骤4 （可选）调用 **CLIENT_SaveRealData** 开始保存预览数据。
- 步骤5 （可选）调用 **CLIENT_StopSaveRealData** 结束保存，生成本地视频文件。
- 步骤6 实时预览使用完毕后，调用 **CLIENT_StopRealPlayEx** 停止实时预览。
- 步骤7 业务使用完后，调用 **CLIENT_Logout** 登出设备。
- 步骤8 SDK 功能使用完后，调用 **CLIENT_Cleanup** 释放 SDK 资源。

注意事项

- 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：接口内申请预览资源需和设备做一些约定，然后才请求预览数据，过程中有一些超时时间的设定（请参见 NET_PARAM 结构体），其中与预览相关的字段为 **nGetConnInfoTime**。如果实际使用中（如网络状况不良）有超时现象，可将 **nGetConnInfoTime** 的值修改大一些。示例代码如下，在 CLIENT_Init 函数后调用，调用一次即可：

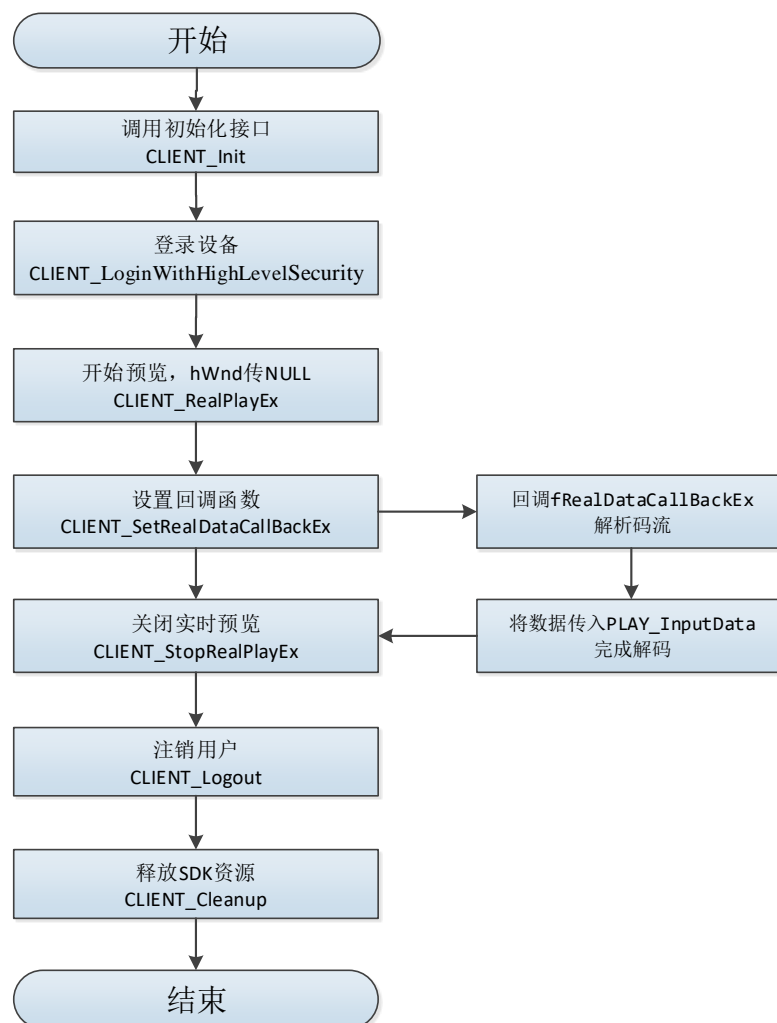
```
NET_PARAM stuNetParam = new NET_PARAM();  
stuNetParam.nGetConnInfoTime = 5000; 为 0 默认为 2*1024*1024  
CLIENT_SetNetworkParam (stuNetParam);
```

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

2.3.3.2 调用私有播放库

SDK 回调实时预览码流给用户，用户调用 PlaySDK 进行解码播放。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_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 在回调函数中将数据传给 `PlaySDK` 完成解码。
- 步骤6 实时预览使用完毕后，调用 `CLIENT_StopRealPlayEx` 停止实时预览。
- 步骤7 业务使用完后，调用 `CLIENT_Logout` 登出设备。
- 步骤8 SDK 功能使用完后，调用 `CLIENT_Cleanup` 释放 SDK 资源。

注意事项

- 码流格式：推荐使用 `PlaySDK` 解码。
- 画面卡顿：
 - ◇ 使用 `PlaySDK` 解码时，解码通道缓存大小有默认（`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出下一段视频数据，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 解码播放

```
import java.awt.Panel;

import main.java.com.netsdk.lib.NetSDKLib.LLong;
import main.java.com.netsdk.lib.ToolKits;

import com.sun.jna.Native;

/**
 * 实时预览接口实现
 * 主要有：开始拉流、停止拉流功能
 */
public class RealPlayModule {
    // 开始预览
    public static LLong startRealPlay(int channel, int stream, Panel realPlayWindow) {
        LLong m_hPlayHandle =
LoginModule.netsdk.CLIENT_RealPlayEx(LoginModule.m_hLoginHandle, channel,
Native.getComponentPointer(realPlayWindow), stream);

        if(m_hPlayHandle.longValue() == 0) {
            System.err.println("开始实时预览失败，错误码" + ToolKits.getErrorCodePrint());
        } else {
            System.out.println("Success to start realplay");
        }

        //自行定义码流保存文件，可选操作，如果需要保存视频则使用
String outFile="example/outputfile";
LoginModule.netsdk.CLIENT_SaveRealData(m_hPlayHandle,outFile);
    }
    return m_hPlayHandle;
}

//停止预览
public static void stopRealPlay(LLong m_hPlayHandle) {
    if(m_hPlayHandle.longValue() == 0) {
        return;
    }
}

//关闭文件保存。
```

```

LoginModule.netsdk.CLIENT_StopSaveRealData(m_hPlayHandle);
    boolean bRet = LoginModule.netsdk.CLIENT_StopRealPlayEx(m_hPlayHandle);
    if(bRet) {
        m_hPlayHandle.setValue(0);
    }
}
}
}

```

2.3.4.2 调用播放库

```

public class RealPlayModule {
    class DataCallBackEx implements NetSDKLib.fRealDataCallBackEx{
        @Override
        public void invoke(LLong lRealHandle, int dwDataType, Pointer pBuffer,
            int dwBufSize, int param, Pointer dwUser) {
            // TODO
            // 从设备获取的码流数据，需调用 PlaySDK 的接口，详见 SDK 预览 demo 源码

        }
    }

    private DataCallBackEx m_DataCallBackEx = new DataCallBackEx();
    public LLong startRealPlay(int channel, int stream, Panel realPlayWindow) {
        LLong m_hPlayHandle =
LoginModule.netsdk.CLIENT_RealPlayEx(LoginModule.m_hLoginHandle, channel,
Native.getComponentPointer(realPlayWindow), stream);

        LoginModule.netsdk.CLIENT_SetRealDataCallBackEx(m_hPlayHandle,m_DataCallBack
Ex, null, 0x00000001);

        if(m_hPlayHandle.longValue() == 0) {
            System.err.println("开始实时预览失败， 错误码" + ToolKits.getErrorCodePrint());
        } else {
            System.out.println("Success to start realplay");
        }

        return m_hPlayHandle;
    }

    public void stopRealPlay(LLong m_hPlayHandle) {
        if(m_hPlayHandle.longValue() == 0) {

```

```
        return;
    }
    boolean bRet = LoginModule.netsdk.CLIENT_StopRealPlayEx(m_hPlayHandle);
    if(bRet) {
        m_hPlayHandle.setValue(0);
    }
}
}
```

2.4 抓图

2.4.1 简介

抓图，即获取前端产品当前画面的图片数据。本章介绍以下抓图方式：

- 同步抓图：用户调用 SDK 接口，接口内部发送抓图命令给设备，设备抓取当前画面并通过网络发送给 SDK，SDK 将接收到的图片数据返回给用户。
- 异步抓图：用户调用 SDK 接口，设置抓图回调，抓到的图片数据在回调函数中体现。同时调用异步抓图接口触发抓图。
- 本地抓图：用户在已打开预览的前提下，可以将监控中的数据保存为图片，该图片是从显示画面中保存帧信息，与设备之间没有网络交互。

2.4.2 接口总览

表2-5 视频抓图的接口信息

接口	说明
CLIENT_SnapPictureToFile	抓图同步接口，将图片数据直接返回给用户。
CLIENT_CapturePictureEx	本地抓图，参数可以是预览或回放的句柄。
CLIENT_SetSnapRevCallBack	设置抓图回调，回调实现 fSnapRev 接口。
CLIENT_SnapPictureEx	异步抓图，适用 IPC、球机等非智能交通、停车场设备。

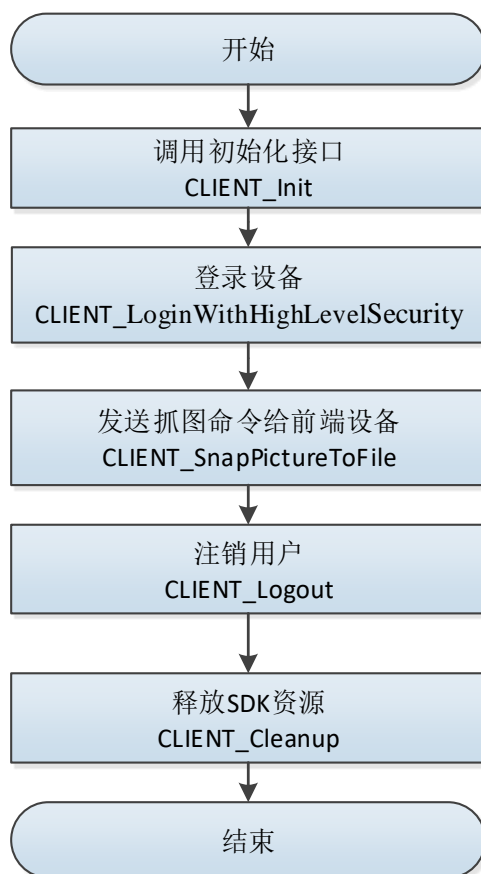
2.4.3 流程说明

抓图可分为同步方式抓图，异步方式抓图和本地抓图。

2.4.3.1 同步抓图

同步抓图流程如图 2-5 所示。

图2-5 同步抓图流程图



流程说明

- 步骤1 调用 `CLIENT_Init` 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 `CLIENT_LoginWithHighLevelSecurity` 登录设备。
- 步骤3 调用 `CLIENT_SnapPictureToFile` 获取图片信息。
- 步骤4 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

- 图片大小限制：SDK 内部分配了固定大小的内存来接收设备端返回的图片数据，如果图片超过默认分配的大小，SDK 返回截断后的图片数据。
SDK 提供了修改默认内存大小的接口，如果实际使用中有图片（如图像分辨率很高）被截断的现象，可以将 `nPicBufSize` 的值修改大一些。示例代码如下，在 `CLIENT_Init` 函数后调用，调用一次即可：

```
NET_PARAM stuNetParam = new NET_PARAM();
stuNetParam.nPicBufSize = 4000*1024*1024; nPicBufSize 默认为 2M
CLIENT_SetNetworkParam (stuNetParam);
```
- 多线程调用：同一个登录会话内的业务，不支持多线程调用。
- 抓图配置：网络抓图的图片质量、分辨率等均支持设置。但是如果默认配置效果能满足，建议不做修改。
- 图片保存形式：图片数据以内存的形式返回，同时接口支持保存成文件（前提是用户已设置 `NET_IN_SNAP_PIC_TO_FILE_PARAM` 的 `szFilePath` 字段）。

2.4.3.2 本地抓图

本地抓图流程如图 2-6 所示。

图2-6 本地抓图流程图



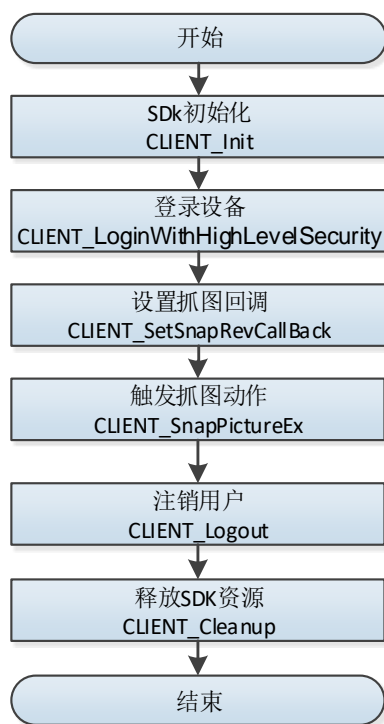
流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginWithHighLevelSecurity 登录设备。
- 步骤3 调用 CLIENT_RealPlayEx 开启预览，获取预览句柄。
- 步骤4 调用 CLIENT_CapturePictureEx 传入预览句柄。
- 步骤5 调用 CLIENT_StopRealPlayEx 关闭实时预览。
- 步骤6 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。

2.4.3.3 异步抓图

本地抓图流程如图 2-6 所示。

图2-7 异步抓图流程图



流程说明

- 步骤1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 CLIENT_LoginWithHighLevelSecurity 登录设备。
- 步骤3 调用 CLIENT_SetSnapRevCallBack 设置抓图回调。
- 步骤4 调用 CLIENT_SnapPictureEx 触发抓图动作，在回调函数中解析抓取到的图片数据。
- 步骤5 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。

2.4.4 示例代码

```
/*
 * 同步抓图示例
 */
NetSDKLib.NET_IN_SNAP_PIC_TO_FILE_PARAM snapParamIn= new
NetSDKLib.NET_IN_SNAP_PIC_TO_FILE_PARAM();
NetSDKLib.NET_OUT_SNAP_PIC_TO_FILE_PARAM snapParamOut= new
NetSDKLib.NET_OUT_SNAP_PIC_TO_FILE_PARAM(1024 * 1024);
snapParamIn.stuParam.Channel = 0;
snapParamIn.stuParam.Quality = 3;
snapParamIn.stuParam.ImageSize = 1; // 0: QCIF,1: CIF,2: D1
snapParamIn.stuParam.mode = 0; // -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
snapParamIn.stuParam.InterSnap = 5;
snapParamIn.stuParam.CmdSerial = serialNum;
```

```

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyMMddHHmmss");
final String fileName = "SyncSnapPicture_" + dateFormat.format(new Date()) + "_" +
serialNum + ".jpg";
System.arraycopy(fileName.getBytes(),0,snapParamIn.szFilePath,0,
fileName.getBytes().length);

    final int timeOut = 5000; // 5 second
    Pointer plnbuf =new Memory(snapParamIn.size());
    plnbuf.clear(snapParamIn.size());
    ToolKits.SetStructDataToPointer(snapParamIn, plnbuf, 0);
    Pointer pOutbuf =new Memory(snapParamOut.size());
    pOutbuf.clear(snapParamOut.size());
    ToolKits.SetStructDataToPointer(snapParamOut, pOutbuf, 0);
if (!netsdkApi.CLIENT_SnapPictureToFile(loginHandle, plnbuf, pOutbuf, timeOut)) {
    System.err.printf("CLIENT_SnapPictureEx Failed! Last Error[%x]\n",
netsdkApi.CLIENT_GetLastError());
    }else {
        System.out.println("CLIENT_SnapPictureToFile Success. " + new
File(fileName).getAbsolutePath());
    }
    Native.free(Pointer.nativeValue(plnbuf));//清理内存
    Pointer.nativeValue(plnbuf, 0); //防止 gc 重复回收

    Native.free(Pointer.nativeValue(pOutbuf));
    Pointer.nativeValue(pOutbuf, 0);

/*
*本地抓图示例
*/
//实时预览
int playType = NetSDKLib.NET_RealPlayType.NET_RType_Realplay; // 实时预览
m_hRealPlayHandle = netsdkApi.CLIENT_RealPlayEx(m_hLoginHandle, channel,
Native.getComponentPointer(realplayPanel), playType);
if (m_hRealPlayHandle.longValue() == 0) {
    System.err.println("开始实时预览失败， 错误码： " + ToolKits.getErrorCode());
    return false;
} else {
    System.out.println("Success to start realplay");
}
//本地抓图
if (!LoginModule.netsdk.CLIENT_CapturePictureEx(hPlayHandle, picFileName,
NetSDKLib.NET_CAPTURE_FORMATS.NET_CAPTURE_JPEG))

```

```

{
    System.err.printf("CLIENT_CapturePicture Failed!" +
ToolKits.getErrorCodePrint());
} else {
    System.out.println("CLIENT_CapturePicture success");
}

//退出预览
if(m_hRealPlayHandle.longValue() != 0) {
    netsdkApi.CLIENT_StopRealPlayEx(m_hRealPlayHandle);
}

/*
 * 异步抓图示例
 */

    /// 设置抓图回调: 图片主要在 SnapCallback.getInstance() invoke. 中返回
    netsdkApi.CLIENT_SetSnapRevCallBack(SnapCallback.getInstance(), null);

    NetSDKLib.SNAP_PARAMS snapParam = new NetSDKLib.SNAP_PARAMS();
    snapParam.Channel = 0; //抓图通道
    snapParam.mode = 0; //表示请求一帧
    snapParam.CmdSerial = serialNum ++; // 请求序列号, 有效值范围 0~65535, 超过
范围会被截断

    /// 触发抓图动作
    if (!netsdkApi.CLIENT_SnapPictureEx(loginHandle, snapParam , null)) {
System.err.printf("CLIENT_SnapPictureEx Failed ! Last Error[%x]\n",
netsdkApi.CLIENT_GetLastError());
        return;
    }

    // 以下保证图片数据的生成
    try {
        synchronized (SnapCallback.class) {
            SnapCallback.class.wait(3000L); // 默认等待 3s, 防止设备断线时抓拍回调
            没有被触发, 而导致死等
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    System.out.println("--> " + Thread.currentThread().getName() + " CLIENT_SnapPictureEx

```

2.5 云台控制

2.5.1 简介

云台是指承载摄像设备及防护罩并能够远程进行全方位控制的机械平台。云台实质上是由两个电机组成，可以实现水平和垂直的运动，从而给摄像机设备全方位、多角度的视野。

本节主要指导用户如何通过 **SDK** 实现方向控制（简称八方位控制，具体包括上、下、左、右，左上、左下、右上和右下）、聚焦、变倍、光圈、快速定位和精确三维定位功能。

CLIENT_DHPTZControlEx 为云台控制基础接口，CLIENT_DHPTZControlEx2 为扩展接口支持的功能更多，两者用法相似，后者参数有所添加。

2.5.2 接口总览

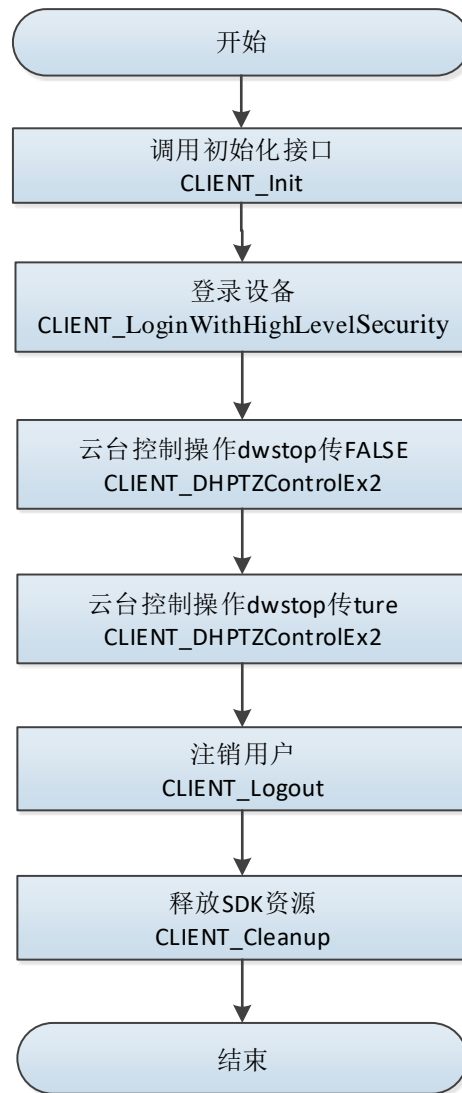
表2-6 云台控制接口信息

接口	说明
CLIENT_DHPTZControlEx	云台控制扩展接口
CLIENT_DHPTZControlEx2	云台控制扩展接口（扩展接口）

2.5.3 流程说明

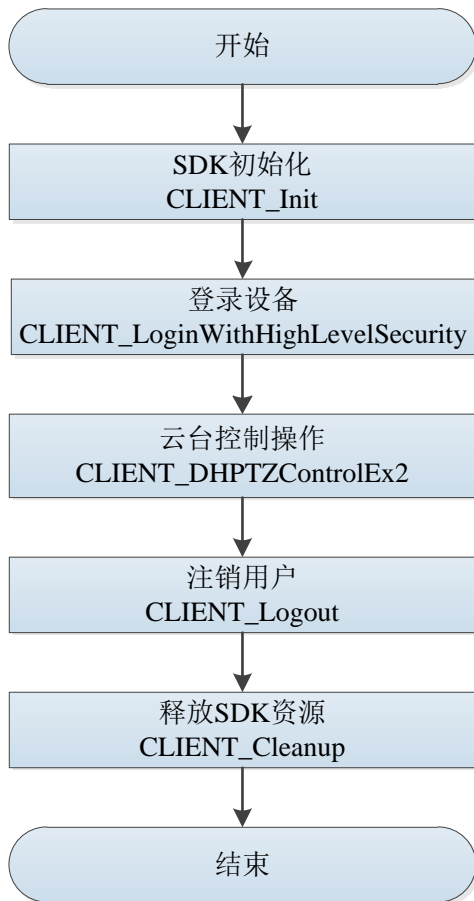
方向控制、聚焦、变倍和光圈属于持续的动作，**SDK** 提供启动和停止的接口，由用户自行控制启动停止的时机。流程如图 2-8 所示。

图2-8 云台控制流程图



快速定位与精确三维定位属于一次动作，只需调用一次云台控制接口，流程如图 2-9 所示。

图2-9 云台控制流程图（一次性操作）



流程说明

- 步骤1 调用 `CLIENT_Init` 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 `CLIENT_LoginWithHighLevelSecurity` 登录设备。
- 步骤3 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，具体请参见 2.5.4 示例代码。
- 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

- 快速定位：以球机当前预览画面中心为原点，水平坐标和垂直坐标有效范围均为[-8191, 8191]。例如传入水平坐标 2000，垂直坐标 2000，球机将向右上方转动并以新的点为坐标原点，即每次指定的坐标都是相对与当前位置的。
- 三维精确定位：首先球机有一个初始位置，以角度为单位，水平坐标[0, 3600]，垂直坐标[-1800, 1800]，每次指定的坐标都是绝对的坐标，和前一次球机画面所处位置无关。
- 更多示例源码见官网发布包（`NetSDK_Chn_java\src\main\java\com\netsdk\demo\frame\PTZControl.java`）。

2.5.4 示例代码

```
/**
```



```

* 云台控制接口实现
* 主要有：八个方向控制、变倍、变焦、光圈功能
*/
public class PtzControlModule {
    /**
     * 向左上
     */
    public static boolean ptzControlLeftUpStart(NetSDKLib.LLong m_hLoginHandle, int
nChannelID, int IParam1, int IParam2) {
        return NetSdk.CLIENT_DHPTZControlEx2(m_hLoginHandle, nChannelID,
NetSDKLib.NET_EXTPTZ_ControlType.NET_EXTPTZ_LEFTTOP,
IParam1, IParam2, 0, 0, null);
    }
    public static boolean ptzControlLeftUpEnd(NetSDKLib.LLong m_hLoginHandle, int
nChannelID) {
        return NetSdk.CLIENT_DHPTZControlEx2(m_hLoginHandle, nChannelID,
NetSDKLib.NET_EXTPTZ_ControlType.NET_EXTPTZ_LEFTTOP,
0, 0, 0, 1, null);
    }
}
//其他功能实现与向上一致都是调用 CLIENT_DHPTZControlEx2 方法，只是入参的类型
NetSDKLib.NET_PTZ_ControlType 不同
}

```

2.6 语音对讲

2.6.1 简介

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

本章主要介绍用户如何使用 SDK 实现与前端设备的语音对讲。

2.6.2 接口总览

表2-7 语音对讲接口信息

接口	说明
CLIENT_StartTalkEx	打开语音对讲扩展接口
CLIENT_StopTalkEx	停止语音对讲扩展接口
CLIENT_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）

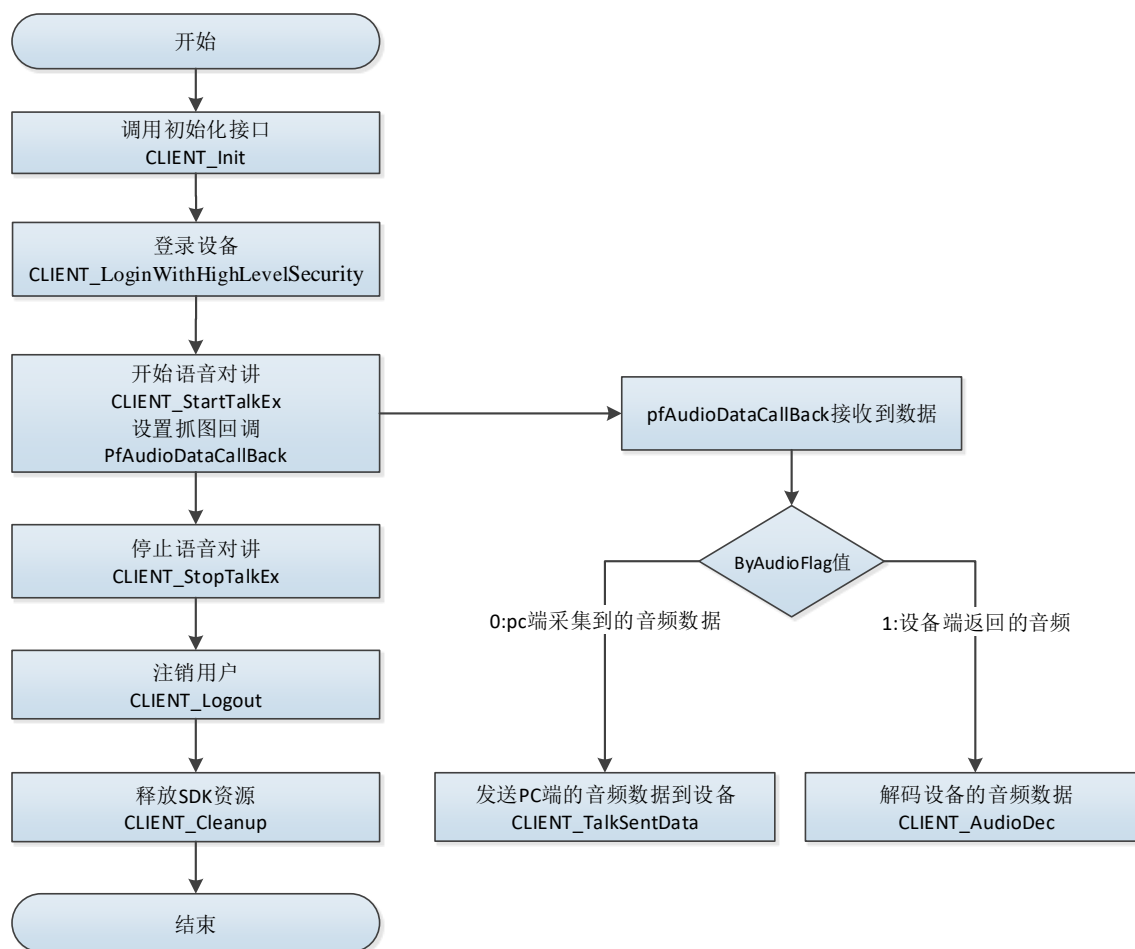
2.6.3 流程说明

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

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

该模式只在 Windows 平台下有效。流程如图 2-10 所示。

图2-10 语音对讲流程图



流程说明

- 步骤1 调用 `CLIENT_Init` 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 `CLIENT_LoginWithHighLevelSecurity` 登录设备。
- 步骤3 调用 `CLIENT_SetDeviceMode` 设置语音对讲编码信息，参数 `emType` 设置为 `DH_TALK_ENCODE_TYPE`。
- 步骤4 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5 调用 `CLIENT_StopTalkEx` 停止语音对讲。
- 步骤6 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

- 语音编码格式：示例采用了常用的 PCM 格式，SDK 支持获取设备支持的语音编码格式，示例源码详见官网发布包
(`NetSDK_Chn_java\src\main\java\com\netsdk\demo\frame\Talk.java`)。
如果默认 PCM 能满足需求，建议不用获取设备支持的语音编码格式。

- 设备端无声音：需要从麦克风等设备采集音频数据，建议检查是否插上麦克风等音频采集设备，同时检查 `CLIENT_RecordStartEx` 接口是否返回成功。

2.6.4 示例代码

```
/**
 * 语音对讲接口实现
 * 包含: 开始通话、结束通话、语音对讲的数据回调实现类
 * \endif
 */
public class TalkModule {
    public static LLong m_hTalkHandle = new LLong(0); // 语音对讲句柄
    private static boolean m_bRecordStatus = false; // 是否正在录音

    /**
     * 开始通话
     */
    public static boolean startTalk(int transferType, int chn) {
        // 设置语音对讲编码格式
        NetSDKLib.NETDEV_TALKDECODE_INFO talkEncode = new
NetSDKLib.NETDEV_TALKDECODE_INFO();
        talkEncode.encodeType =
NetSDKLib.NET_TALK_CODING_TYPE.NET_TALK_PCM;
        talkEncode.dwSampleRate = 8000;
        talkEncode.nAudioBit = 16;
        talkEncode.nPacketPeriod = 25;
        talkEncode.write();
        if(LoginModule.netsdk.CLIENT_SetDeviceMode(LoginModule.m_hLoginHandle,
NetSDKLib.EM_USEDEV_MODE.NET_TALK_ENCODE_TYPE, talkEncode.getPointer())) {
            System.out.println("Set Talk Encode Type Succeed!");
        } else {
            System.err.println("Set Talk Encode Type Failed!" +
ToolKits.getErrorCodePrint());
            return false;
        }
        // 设置语音对讲喊话参数
        NetSDKLib.NET_SPEAK_PARAM speak = new NetSDKLib.NET_SPEAK_PARAM();
        speak.nMode = 0;
        speak.bEnableWait = false;
        speak.nSpeakerChannel = 0;
        speak.write();
        if (LoginModule.netsdk.CLIENT_SetDeviceMode(LoginModule.m_hLoginHandle,
```

```

NetSDKLib.EM_USEDEV_MODE.NET_TALK_SPEAK_PARAM, speak.getPointer())) {
    System.out.println("Set Talk Speak Mode Succeed!");
} else {
    System.err.println("Set Talk Speak Mode Failed!" + ToolKits.getErrorCodePrint());
    return false;
}
// 设置语音对讲是否为转发模式
NetSDKLib.NET_TALK_TRANSFER_PARAM talkTransfer = new
NetSDKLib.NET_TALK_TRANSFER_PARAM();
talkTransfer.bTransfer = transferType;
talkTransfer.write();
if(LoginModule.netsdk.CLIENT_SetDeviceMode(LoginModule.m_hLoginHandle,
NetSDKLib.EM_USEDEV_MODE.NET_TALK_TRANSFER_MODE, talkTransfer.getPointer()))
{
    System.out.println("Set Talk Transfer Mode Succeed!");
} else {
    System.err.println("Set Talk Transfer Mode Failed!" +
ToolKits.getErrorCodePrint());
    return false;
}
if (talkTransfer.bTransfer == 1) { // 转发模式设置转发通道
    IntByReference nChn = new IntByReference(chn);
    if(LoginModule.netsdk.CLIENT_SetDeviceMode(LoginModule.m_hLoginHandle,
NetSDKLib.EM_USEDEV_MODE.NET_TALK_TALK_CHANNEL, nChn.getPointer())) {
        System.out.println("Set Talk Channel Succeed!");
    } else {
        System.err.println("Set Talk Channel Failed!" + ToolKits.getErrorCodePrint());
        return false;
    }
}
m_hTalkHandle =
LoginModule.netsdk.CLIENT_StartTalkEx(LoginModule.m_hLoginHandle,
AudioDataCB.getInstance(), null);
if(m_hTalkHandle.longValue() == 0) {
    System.err.println("Start Talk Failed!" + ToolKits.getErrorCodePrint());
    return false;
} else {
    System.out.println("Start Talk Success");
    if(LoginModule.netsdk.CLIENT_RecordStart()){
        System.out.println("Start Record Success");
        m_bRecordStatus = true;
    }
}

```

```

        } else {
            System.err.println("Start Local Record Failed!" +
ToolKits.getErrorCodePrint());
            stopTalk();
            return false;
        }
    }
    return true;
}
/**
 * 结束通话
 */
public static void stopTalk() {
    if(m_hTalkHandle.longValue() == 0) {
        return;
    }

    if (m_bRecordStatus){
        LoginModule.netsdk.CLIENT_RecordStop();
        m_bRecordStatus = false;
    }
    if(LoginModule.netsdk.CLIENT_StopTalkEx(m_hTalkHandle)) {
        m_hTalkHandle.setValue(0);
    }else {
        System.err.println("Stop Talk Failed!" + ToolKits.getErrorCodePrint());
    }
}
/**
 * 语音对讲的数据回调
 */
private static class AudioDataCB implements NetSDKLib.pfAudioDataCallBack {

    private AudioDataCB() {}
    private static AudioDataCB audioCallBack = new AudioDataCB();
    public static AudioDataCB getInstance() {
        return audioCallBack;
    }

    public void invoke(LLong lTalkHandle, Pointer pDataBuf, int dwBufSize, byte
byAudioFlag, Pointer dwUser){
        if(lTalkHandle.longValue() != m_hTalkHandle.longValue()) {

```

```
        return;
    }
    if (byAudioFlag == 0) { // 将收到的本地 PC 端检测到的声卡数据发送给设备端

        LLong ISendSize =
LoginModule.netsdk.CLIENT_TalkSendData(m_hTalkHandle, pDataBuf, dwBufSize);
        if(ISendSize.longValue() != (long)dwBufSize) {
            System.err.println("send incomplete" + ISendSize.longValue() + ":" +
dwBufSize);
        }
    }else if (byAudioFlag == 1) { // 将收到的设备端发送过来的语音数据传给 SDK 解
码播放
        LoginModule.netsdk.CLIENT_AudioDecEx(m_hTalkHandle, pDataBuf,
dwBufSize);
    }
}
}
```

2.7 报警监听

2.7.1 简介

智能设备对实时码流进行分析，当检测到预先设定好的事件时，进行报警处理。

2.7.2 接口总览

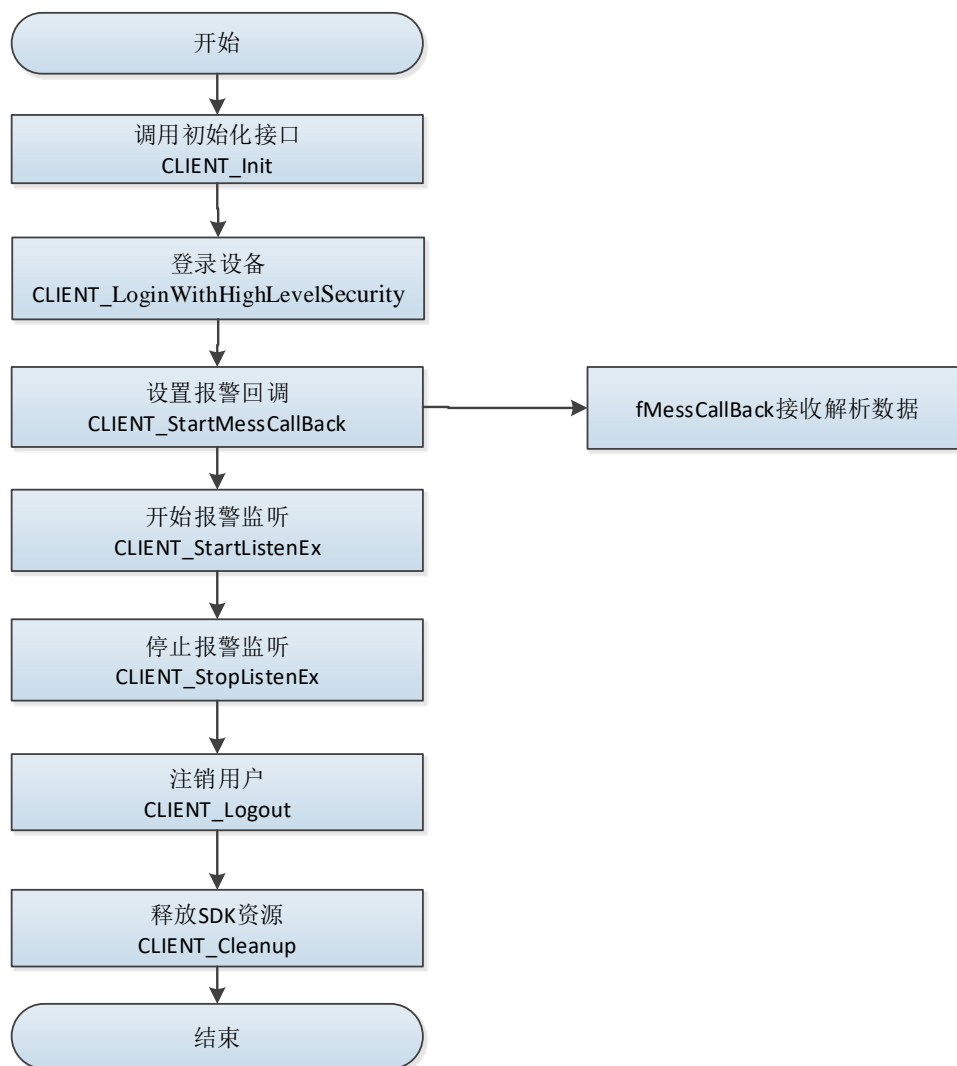
表2-8 报警监听接口信息

接口	说明
CLIENT_StartListenEx	向设备订阅报警--扩展
CLIENT_StopListen	停止订阅报警
CLIENT_SetDVRMessCallBack	设置报警监听

2.7.3 流程说明

流程如图 2-11 所示。

图2-11 报警监听流程图



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，设置报警回调，调用 `CLIENT_SetDVRMessCallBack` 方法。
- 步骤4 调用 `CLIENT_StartListenEx` 开始报警监听。
- 步骤5 监听成功后设备上报的报警事件通过 `MessCallBack` 回调函数获取报警事件并通知用户。
- 步骤6 使用完毕后调用 `CLIENT_StopListen` 停止报警监听。
- 步骤7 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

无数据上报：调用 `CLIENT_SetDVRMessCallBack` 才会开始获取报警事件数据。

2.7.4 示例代码

```
// 设置报警回调
netSdk.CLIENT_SetDVRMessCallBack(callback, null);
//报警监听
    if (listening) {
        return true;
    }
    listening = netSdk.CLIENT_StartListenEx(loginHandle);
    if (!listening) {
        System.err.println("Start Listen Failed!" + ToolKits.getErrorCode());
    } else {
        System.out.println("Start Listen Success.");
    }
}
//fmessCallback 回调函数
public class MessCallBack implements NetSDKLib.fMessCallBack {
    private MessCallBack() {}
    private static class CallBackHolder {
        private static final MessCallBack cb = new MessCallBack();
    }

    public static final MessCallBack getInstance() {
        return CallBackHolder.cb;
    }
    @Override
    public boolean invoke(int ICommand, LLong ILoginID, Pointer pStuEvent,
        int dwBufLen, String strDeviceIP, NativeLong nDevicePort,
        Pointer dwUser) {
        switch (ICommand) {
            case NetSDKLib.NET_ALARM_ACCESS_CTL_EVENT : // 门禁事件
            {
                ALARM_ACCESS_CTL_EVENT_INFO msg = new
ALARM_ACCESS_CTL_EVENT_INFO();
                ToolKits.GetPointerData(pStuEvent, msg);
                System.out.println("【门禁事件】 " + msg);
                break;
            }
        }
    }
}
//停止报警监听
if (listening) {
```



```
netSdk.CLIENT_StopListen(loginHandle);  
listening = false;  
}
```

2.8 智能订阅

2.8.1 简介

智能订阅，即智能设备对实时码流进行分析，当检测到预先设定好的事件时，将事件发送给用户。智能事件有交通违章、停车场有无车位等事件。

智能订阅实现方式为 SDK 主动连接设备，并向设备订阅智能事件功能，设备检测到智能事件立即发送给 SDK。

当前支持的智能订阅事件参见 NetSDKLib.java 中以 EVENT_IVS_开头的常量，包含了常规的交通占道、车辆违规等事件。

2.8.2 接口总览

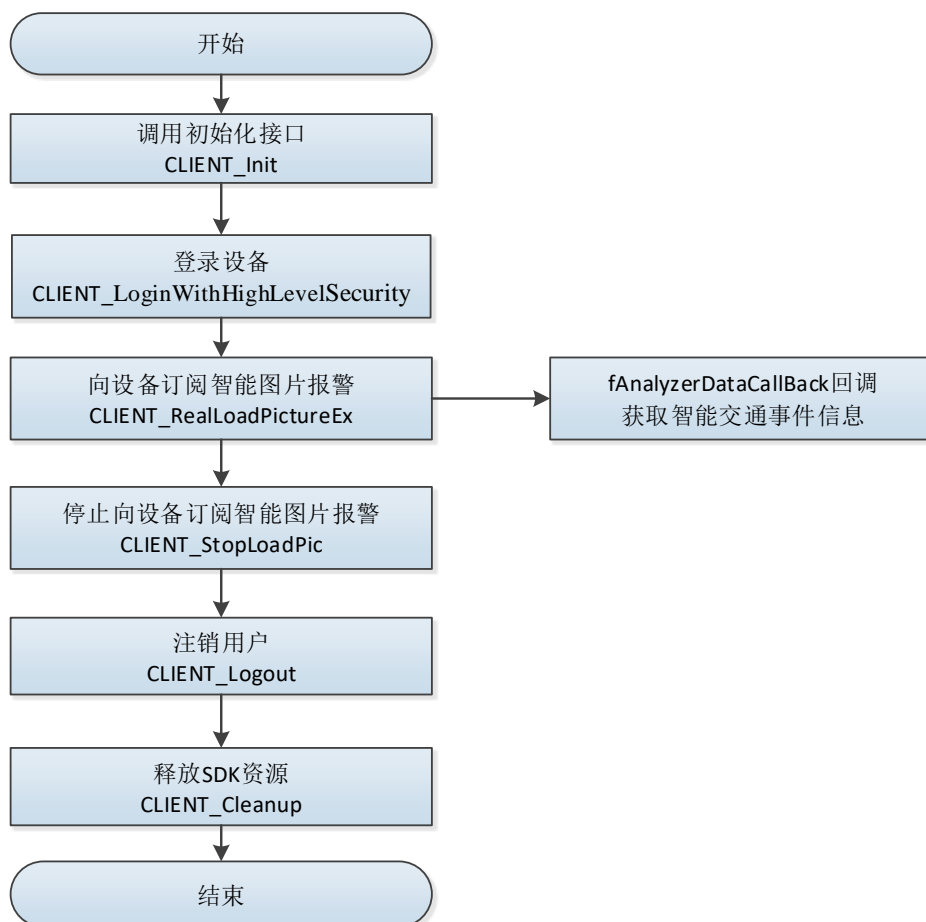
表2-9 智能交通时间上报的接口信息

接口	说明
CLIENT_RealLoadPictureEx	订阅智能事件
CLIENT_StopLoadPic	取消订阅智能事件
fAnalyzerDataCallBack	用于回调获取智能事件的信息

2.8.3 流程说明

智能订阅事件上报流程如图 2-12 所示。

图2-12 智能订阅事件上报业务流程



流程说明

- 步骤1 调用 **CLIENT_Init** 函数完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 **CLIENT_LoginWithHighLevelSecurity** 函数登录设备。
- 步骤3 调用 **CLIENT_RealLoadPictureEx** 函数向设备订阅智能交通事件。
- 步骤4 订阅成功后设备上报的智能交通事件通过 **fAnalyzerDataCallBack** 回调函数获取智能交通事件并通知用户。
- 步骤5 智能交通事件上报功能使用完毕后，调用 **CLIENT_StopLoadPic** 函数停止订阅智能交通事件。
- 步骤6 业务使用完后，调用 **CLIENT_Logout** 函数登出设备。
- 步骤7 SDK 功能使用完后，调用 **CLIENT_Cleanup** 函数释放 SDK 资源。

注意事项

- 订阅事件类型：如果需要同时上报不同智能事件时，支持订阅所有智能事件（**EVENT_IVS_ALL**）；也支持订阅单个智能事件。
- 设置是否接收图片：由于某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**，只接收智能交通事件信息，不带图片。

2.8.4 示例代码

```
//本实例将用门禁事件举例
//省略 SDK 初始化以及门禁设备登录
// 智能订阅句柄
private LLong attachHandle = new NetSDKLib.LLong(0);
/**
 * 订阅智能任务
 */
public void AttachEventRealLoadPic() {
    // 先退订，设备不会对重复订阅作校验，重复订阅后会有重复的事件返回
    this.DetachEventRealLoadPic();
    // 需要图片
    int bNeedPicture = 1;
    attachHandle = netsdkApi.CLIENT_RealLoadPictureEx(loginHandle, channel,
NetSDKLib.EVENT_IVS_ALL, bNeedPicture,
        AnalyzerDataCB.getInstance(), null, null);
    if (attachHandle.longValue() != 0) {
        System.out.printf("Chn[%d] CLIENT_RealLoadPictureEx Success\n", channel);
    } else {
        System.out.printf("Ch[%d] CLIENT_RealLoadPictureEx Failed!LastError = %s\n",
channel, ToolKits.getErrorCode());
    }
}
/**
 * 取消订阅
 */
public void DetachEventRealLoadPic() {
    if (attachHandle.longValue() != 0) {
        netsdkApi.CLIENT_StopLoadPic(attachHandle);
    }
}

//门禁系统智能事件回调，继承自 fAnalyzerDataCallBack 并自己实现逻辑
private static class AnalyzerDataCB implements NetSDKLib.fAnalyzerDataCallBack {
    private final File picturePath;
    private static AnalyzerDataCB instance;
    private AnalyzerDataCB() {
        picturePath = new File("./AnalyzerPicture/");
        if (!picturePath.exists()) {
```

```

        picturePath.mkdirs();
    }
}

public static AnalyzerDataCB getInstance() {
    if (instance == null) {
        synchronized (AnalyzerDataCB.class) {
            if (instance == null) {
                instance = new AnalyzerDataCB();
            }
        }
    }
    return instance;
}

private BufferedImage gateBufferedImage = null;
@Override
public int invoke(LLong lAnalyzerHandle, int dwAlarmType,
    Pointer pAlarmInfo, Pointer pBuffer, int dwBufSize,
    Pointer dwUser, int nSequence, Pointer reserved)
{
    if (lAnalyzerHandle.longValue() == 0 || pAlarmInfo == null) {
        return -1;
    }

    switch(dwAlarmType)
    {
        case NetSDKLib.EVENT_IVS_ACCESS_CTL: ///< 门禁事件
        {
            DEV_EVENT_ACCESS_CTL_INFO msg = new DEV_EVENT_ACCESS_CTL_INFO();
            ToolKits.GetPointerData(pAlarmInfo, msg);

            System.out.println("事件名称 : " + new String(msg.szName).trim());
            if(msg.emEventType == 1) {
                System.out.println("门禁事件类型 : 进门! ");
            } else if(msg.emEventType == 2){
                System.out.println("门禁事件类型 : 出门! ");
            }

            if(msg.bStatus == 1) {
                System.out.println("刷卡结果 : 成功! ");
            }
        }
    }
}

```

```

    } else if(msg.bStatus == 0) {
        System.out.println("刷卡结果 : 失败!");
    }

    System.out.println("卡类型: " + msg.emCardType);
    System.out.println("开门方式: " + msg.emOpenMethod);
    System.out.println("卡号 : " + new String(msg.szCardNo).trim());
    System.out.println("开门用户 : " + new String(msg.szUserID).trim());
    System.out.println("开门失败原因错误码: " + msg.nErrorCode);
    System.out.println("考勤状态: " + msg.emAttendanceState);
    System.out.println("卡命名 : " + new String(msg.szCardName).trim());

    try {
        System.out.println("角色:" + new String(msg.stuCustomWorkerInfo.szRole, "GBK").trim());
        System.out.println("项目编号:" + new String(msg.stuCustomWorkerInfo.szProjectNo).trim());
        System.out.println("项目名称:" + new String(msg.stuCustomWorkerInfo.szProjectName,
            "GBK").trim());
        System.out.println("施工单位全称:" + new String(msg.stuCustomWorkerInfo.szBuilderName,
            "GBK").trim());
    } catch (UnsupportedEncodingException e) {
        System.err.println("...UnsupportedEncodingException...");
    }

    if (msg.nImageInfoCount == 0) {
        // 抓拍图片, 设备只返回一个抓图
        String snapPicPath = path + "\\" + System.currentTimeMillis() +
            "AccessSnapPicture.jpg"; // 保存图片地址
        byte[] buffer = pBuffer.getByteArray(0, dwBufSize);
        ByteArrayInputStream byteArrInputGlobal = new
        ByteArrayInputStream(buffer);
        try {
            BufferedImage bufferedImage = ImageIO.read(byteArrInputGlobal);
            if(bufferedImage != null) {
                ImageIO.write(bufferedImage, "jpg", new File(snapPicPath));
                System.out.println("抓拍图片保存路径: " + snapPicPath);
            }
        } catch (IOException e2) {
            e2.printStackTrace();
        }
    } else {

```

```

        String snapPicPath;
        for (int i = 0; i < msg.nImageInfoCount; ++i) {
            snapPicPath = path + "\\" + System.currentTimeMillis() + "_AccessSnapPicture_" + i
+ ".jpg"; // 保存图片地址
byte[] buffer=pBuffer.getByteArray(msg.stulImageInfo[i].nOffSet, msg.stulImageInfo[i].nLength);
ByteArrayInputStream byteArrInputGlobal = new ByteArrayInputStream(buffer);
            try {
                BufferedImage bufferedImage = ImageIO.read(byteArrInputGlobal);
                if(bufferedImage != null) {
ImageIO.write(bufferedImage, "jpg", new File(snapPicPath));
System.out.println("抓拍图片保存路径: " + snapPicPath);
                }
            } catch (IOException e2) {
                e2.printStackTrace();
            }
        }
        break;
    }
    default:
        break;
}
}

```

2.9 录像回放

2.9.1 简介

录像回放是有针对地对系统中的一些通道进行特定时间段的视频回放操作，以实现在海量的视频信息中找到目标视频进而进行调查。

回放功能主要包括：开始回放、暂停回放、恢复暂停和停止回放。

2.9.2 接口总览

表2-10 录像回放的接口信息

接口	说明
CLIENT_PlayBackByTimeEx	按时间方式回放扩展接口
CLIENT_SetDeviceMode	设置设备语音对讲、回放、权限等工作模式接口
CLIENT_StopPlayBack	停止录像回放接口

接口	说明
CLIENT_PausePlayBack	暂停或恢复录像回放

2.9.3 流程说明

SDK 初始化之后，用户只需输入通道号、录像的起始时间、结束时间和有效窗口句柄，即可实现所需录像段录像回放。如图 2-13 所示。

图2-13 录像回放流程



流程说明

- 步骤1 调用 `CLIENT_Init` 完成 SDK 初始化流程。
- 步骤2 初始化成功后，调用 `CLIENT_LoginWithHighLevelSecurity` 登录设备。
- 步骤3 （可选）调用两次 `CLIENT_SetDeviceMode`，设置回放时的码流类型参数 `emType` 为 `DH_RECORD_STREAM_TYPE`，回放时的录像文件类型参数 `emType` 为 `DH_RECORD_TYPE`。
- 步骤4 调用 `CLIENT_PlayBackByTimeEx` 开始录像回放，`hWnd` 参数为有效窗口句柄值。
- 步骤5 （可选）调用 `CLIENT_PausePlayBack`，第二个参数为 1 时暂停回放。
- 步骤6 （可选）调用 `CLIENT_PausePlayBack`，第二个参数为 0 时恢复回放。
- 步骤7 回放使用完后，调用 `CLIENT_StopPlayBack` 停止回放。
- 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。

2.9.4 示例代码

```
// 开启回放
private void StartPlayBack() {
    if (m_hLoginHandle.longValue() == 0) {
        System.err.printf("Please Login First");
        return;
    }

    // 同一个登录句柄，回放和下载不能同时进行
    if (m_hDownloadHandle.longValue() != 0) {
        JOptionPane.showMessageDialog(playFrame, "请先停止下载");
        return;
    }

    updatePlayBackParams(); // 更新参数

    // 设置回放时的码流类型
    IntByReference steamType = new IntByReference(m_streamType); // 0-主辅码流,1-主码流,2-辅码流
    int emType = NetSDKLib.EM_USEDEV_MODE.NET_RECORD_STREAM_TYPE;

    boolean bret = NetSdk.CLIENT_SetDeviceMode(m_hLoginHandle, emType,
        steamType.getPointer());
    if (!bret) {
        System.err.printf("Set Stream Type Failed, Get last error [0x%x]\n",
            NetSdk.CLIENT_GetLastError());
    }
}
```



```

        // 设置回放时的录像文件类型
        IntByReference emFileType = new IntByReference(m_recordType); // 所有录像
NET_RECORD_TYPE
        emType = NetSDKLib.EM_USEDEV_MODE.NET_RECORD_TYPE;
        bret = NetSdk.CLIENT_SetDeviceMode(m_hLoginHandle, emType,
emFileType.getPointer());
        if (!bret) {
            System.err.printf("Set Record Type Failed, Get last error [0x%x]\n",
NetSdk.CLIENT_GetLastError());
        }

        m_hPlayHandle = NetSdk.CLIENT_PlayBackByTimeEx(m_hLoginHandle,
m_channel.intValue(), m_startTime, m_stopTime,
            playWindow.getHWNDofFrame(), m_PlayBackDownLoadPos, null,
m_dataCallBack, null);
        if (m_hPlayHandle.longValue() == 0) {
            int error = NetSdk.CLIENT_GetLastError();
            System.err.printf("PlayBackByTimeEx Failed, Get last error [0x%x]\n", error);
            switch(error) {
                case LastError.NET_NO_RECORD_FOUND:
                    JOptionPane.showMessageDialog(playFrame, "查找不到录像");

                    break;
                default:
                    JOptionPane.showMessageDialog(playFrame, "开启失败, 错误码: " +
String.format("0x%x", error));
                    break;
            }
        }
        else {
            System.out.println("PlayBackByTimeEx Succeeded");
            m_playFlag = true; // 打开播放标志位
            playButton.setText("停止回放");
            panelPlayBack.repaint();
            panelPlayBack.setVisible(true);
        }
    }

    //停止回放
    private void StopPlayBack() {

```

```

        if (m_hPlayHandle.longValue() == 0) {
            System.err.println("Please make sure the PlayBack Handle is valid");
            return;
        }

        if (!NetSdk.CLIENT_StopPlayBack(m_hPlayHandle)) {
            System.err.println("StopPlayBack Failed");
            return;
        }

        m_hPlayHandle.setValue(0);
        m_playFlag = false;
        m_pauseFlag = true;
        playPos = 0;
        playButton.setText("开启回放");
        pauseButton.setText("暂停");
        panelPlayBack.repaint();
    }

    /**
     * 回放暂停、播放
     * @param pause true - 暂停; false - 播放
     */
    private void PausePlayBack(boolean pause) {
        if (m_hPlayHandle.longValue() == 0) {
            System.err.println("Please make sure the PlayBack Handle is valid");
            return;
        }

        NetSdk.CLIENT_PausePlayBack(m_hPlayHandle, pause ? 1 : 0); // 1 - 暂停 0 - 恢复
        pauseButton.setText(pause ? "播放":"暂停");
    }

    //常速播放
    private void NormalPlayBack() {
        if (m_hPlayHandle.longValue() == 0) {
            System.err.println("Please make sure the PlayBack Handle is valid");
            return;
        }
    }

```

```

        NetSdk.CLIENT_NormalPlayBack(m_hPlayHandle);
    }

    //快放
    private void FastPlayBack() {
        if (m_hPlayHandle.longValue() == 0) {
            System.err.println("Please make sure the PlayBack Handle is valid");
            return;
        }

        NetSdk.CLIENT_FastPlayBack(m_hPlayHandle);
    }

    //慢放
    private void SlowPlayBack() {
        if (m_hPlayHandle.longValue() == 0) {
            System.err.println("Please make sure the PlayBack Handle is valid");
            return;
        }

        NetSdk.CLIENT_SlowPlayBack(m_hPlayHandle);
    }

```

2.10 录像下载

2.10.1 简介

视频监控系统在城市、机场、地铁、银行和工厂等场合有大量的应用，当事件发生后，常需要下载视频录像给上级领导、公安部门或媒体做进一步应用。因此视频录像的下载是一个非常重要的功能。

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。

录像下载仅支持带有存储功能的录像设备使用。

2.10.2 接口总览

表2-11 录像下载的接口信息

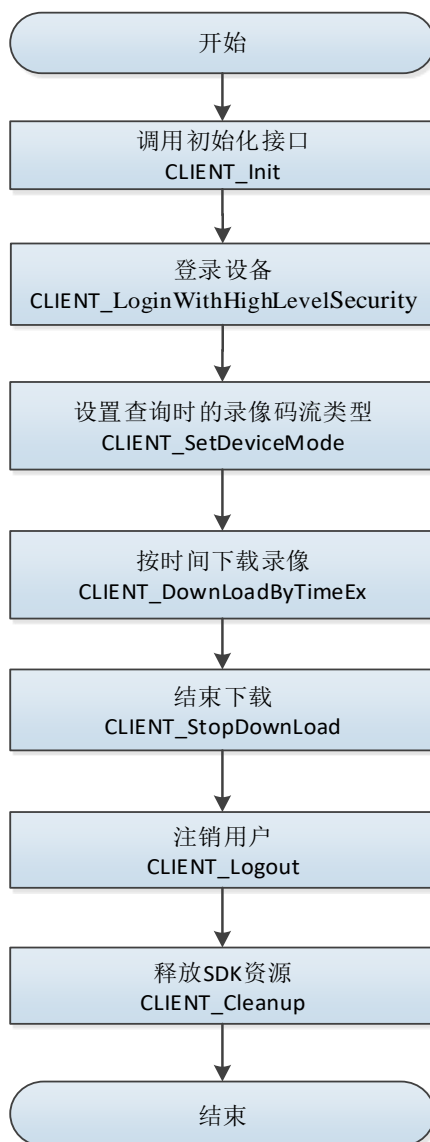
接口	说明
CLIENT_QueryRecordFile	查询时间段内的所有录像文件的接口。
CLIENT_DownloadByTimeEx	按时间下载录像扩展接口。
CLIENT_StopDownload	停止录像下载接口。

2.10.3 流程说明

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。

按时间下载流程如图 2-14 所示。

图2-14 按时间下载流程图



流程说明

- 步骤1 调用 `CLIENT_Init` 函数，完成 SDK 初始化流程。
- 步骤2 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**。
- 步骤4 调用 `CLIENT_DownloadByTimeEx` 接口函数开始按时间下载，形参 **sSavedFileName** 和 **fDownloadDataCallback** 中至少有一个需为有效值。**cbDownloadPos** 用户可自行决定是否使用，如果不适用置为 **NULL** 即可。
- 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

完成后关闭下载，也可以下载一部分就关闭下载。

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。

2.10.4 示例代码

```
import main.java.com.netsdk.lib.NetSDKLib;
import main.java.com.netsdk.lib.NetSDKLib.LLong;
import main.java.com.netsdk.lib.ToolKits;

import com.sun.jna.ptr.IntByReference;

/**
 * 下载录像接口实现
 * 主要有： 查询录像、下载录像、设置码流类型功能
 */
public class DownLoadRecordModule {
    // 下载句柄
    public static LLong m_hDownLoadHandle = new LLong(0);

    // 查找录像文件
    public static boolean queryRecordFile(int nChannelId,
                                           NetSDKLib.NET_TIME stTimeStart,
                                           NetSDKLib.NET_TIME stTimeEnd,
                                           NetSDKLib.NET_RECORDFILE_INFO[] stFileInfo,
                                           IntByReference nFindCount) {
        // RecordFileType 录像类型 0:所有录像 1:外部报警 2:动态监测报警 3:所有报警
        // 4:卡号查询 5:组合条件查询
        // 6:录像位置与偏移量长度 8:按卡号查询图片(目前仅 HB-U 和 NVS 特殊型号的设备支持)
        // 9:查询图片(目前仅 HB-U 和 NVS 特殊型号的设备支持)
        // 10:按字段查询 15:返回网络数据结构(金桥网吧) 16:查询所有透明串数据录像文件
        int nRecordFileType = 0;
        boolean bRet =
LoginModule.netsdk.CLIENT_QueryRecordFile(LoginModule.m_hLoginHandle, nChannelId,
nRecordFileType, stTimeStart, stTimeEnd, null, stFileInfo, stFileInfo.length *
stFileInfo[0].size(), nFindCount, 5000, false);

        if(bRet) {
            System.out.println("QueryRecordFile Succeed! \n" + "查询到的视频个数: " +
nFindCount.getValue());
        }
    }
}
```

```

    } else {
        System.err.println("QueryRecordFile Failed!" + ToolKits.getErrorCodePrint());
        return false;
    }
    return true;
}

/**
 * 设置回放时的码流类型
 * @param m_streamType 码流类型
 */
public static void setStreamType(int m_streamType) {

    IntByReference steamType = new IntByReference(m_streamType);// 0-主辅码流,1-主
    码流,2-辅码流

    int emType = NetSDKLib.EM_USEDEV_MODE.NET_RECORD_STREAM_TYPE;

    boolean bret =
    LoginModule.netsdk.CLIENT_SetDeviceMode(LoginModule.m_hLoginHandle, emType,
    steamType.getPointer());
    if (!bret) {
        System.err.println("Set Stream Type Failed, Get last error." +
        ToolKits.getErrorCodePrint());
    } else {
        System.out.println("Set Stream Type Succeed!");
    }
}

//下载录像
public static LLong downloadRecordFile(int nChannelId,
                                        int nRecordFileType,
                                        NetSDKLib.NET_TIME stTimeStart,
                                        NetSDKLib.NET_TIME stTimeEnd,
                                        String SavedFileName,
                                        NetSDKLib.fTimeDownloadPosCallBack
    cbTimeDownloadPos) {

    m_hDownloadHandle =
    LoginModule.netsdk.CLIENT_DownloadByTimeEx(LoginModule.m_hLoginHandle,
    nChannelId, nRecordFileType, stTimeStart, stTimeEnd, SavedFileName,
    cbTimeDownloadPos, null, null, null, null);
    if(m_hDownloadHandle.longValue() != 0) {

```

```

        System.out.println("Downloading RecordFile!");
    } else {
        System.err.println("Download RecordFile Failed!" +
ToolKits.getErrorCodePrint());
    }
    return m_hDownloadHandle;
}

public static void stopDownloadRecordFile(LLong m_hDownloadHandle) {
    if (m_hDownloadHandle.longValue() == 0) {
        return;
    }
    LoginModule.netsdk.CLIENT_StopDownload(m_hDownloadHandle);
}
}

```

2.11 实时预览转码

2.11.1 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

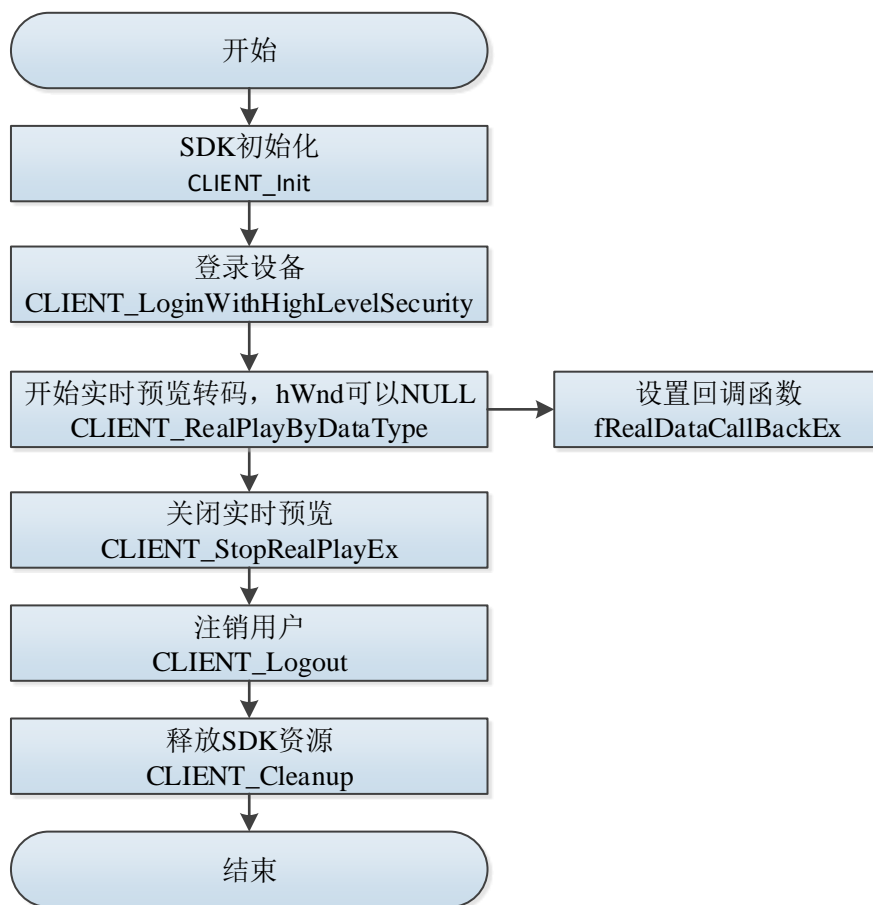
2.11.2 接口总览

表2-12 实时预览转码的接口信息

接口	说明
CLIENT_RealPlayByDataType	开始实时预览转码接口
CLIENT_StopRealPlay	停止实时预览转码接口

2.11.3 流程说明

图2-15 实时预览转码流程



流程说明

实时预览转码流程如图 2-15 所示。

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。

2.11.4 示例代码

```
import com.netsdk.lib.NetSDKLib;
import com.netsdk.lib.NetSDKLib.*;
import com.sun.jna.Native;
import com.sun.jna.Pointer;
```

```

import javax.swing.*;
import java.awt.*;
import java.util.Vector;

public class CommonWithCallBack { // 带回调方法
    static NetSDKLib netsdkApi = NetSDKLib.NETSDK_INSTANCE;
    LLong loginHandle;
    JWindow wnd;
    private static final int MAX_WINDOW_NUM = 4;
    Vector<JWindow> vecWnd;
    public CommonWithCallBack(LLong loginHandle)
    {
        this.loginHandle = loginHandle;
        createWindow();
    }

    /**
     * 裸流回调，此功能所用库，需要开宏
     */
    public void RealPlayByDataType() {

        wnd.setVisible(true);

        NetSDKLib.NET_IN_REALPLAY_BY_DATA_TYPE stIn = new
NetSDKLib.NET_IN_REALPLAY_BY_DATA_TYPE();
        stIn.hWnd = Native.getComponentPointer(wnd);
        stIn.emDataType =
EM_REAL_DATA_TYPE.EM_REAL_DATA_TYPE_FLV_STREAM;
        stIn.nChannelID = 0;
        stIn.rType = NET_RealPlayType.NET_RType_Realplay;
        stIn.cbRealData = RealDataCallBack.getInstance();
        stIn.dwUser = null;
        stIn.szSaveFileName = "d:/123.flv"; // 转换后的裸 H264 码流文件名

        NetSDKLib.NET_OUT_REALPLAY_BY_DATA_TYPE stOut = new
NetSDKLib.NET_OUT_REALPLAY_BY_DATA_TYPE();

        LLong lRealHandle = netsdkApi.CLIENT_RealPlayByDataType(loginHandle, stIn,
stOut, 5000);
        if(lRealHandle.longValue() != 0) {
            System.out.println("RealPlayByDataType Succeed!");
        } else {
            System.err.printf("RealPlayByDataType Failed!Last Error[0x%x]\n",
netsdkApi.CLIENT_GetLastError());
            return;
        }
        try {
            Thread.sleep(10000);

```

```

    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    // 停止预览
    netsdkApi.CLIENT_StopRealPlay(lRealHandle); // 必须停止拉流后，才会生成
123.dat
    wnd.setVisible(false);
}

// 回调建议写成单例模式，回调里处理数据，需要另开线程
public static class RealDataCallBack implements NetSDKLib.fRealDataCallBackEx {
    private RealDataCallBack() {}

    private static class RealDataCallBackHolder {
        private static final RealDataCallBack realDataCB = new RealDataCallBack();
    }

    public static final RealDataCallBack getInstance() {
        return RealDataCallBackHolder.realDataCB;
    }

    @Override
    public void invoke(LLong lRealHandle, int dwDataType,
        Pointer pBuffer, int dwBufSize, int param, Pointer dwUser) {
        System.out.println("RealDataCallBack dwDataType : " + dwDataType);
        if (dwDataType == (NetSDKLib.NET_DATA_CALL_BACK_VALUE +
EM_REAL_DATA_TYPE.EM_REAL_DATA_TYPE_FLV_STREAM)) {
            System.out.println("RealDataCallBack dwDataType : " + dwDataType);
        }
    }
}

public void createWindow() {
    wnd = new JWindow();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    screenSize.height /= 2;
    screenSize.width /= 2;
    wnd.setSize(screenSize);

    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
    int w = wnd.getSize().width;
    int h = wnd.getSize().height;
    int x = (dim.width - w) / 2;
    int y = (dim.height - h) / 2;
    wnd.setLocation(x, y);
}

```

```
}  
}
```

2.12 录像回放转码

2.12.1 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。

2.12.2 接口总览

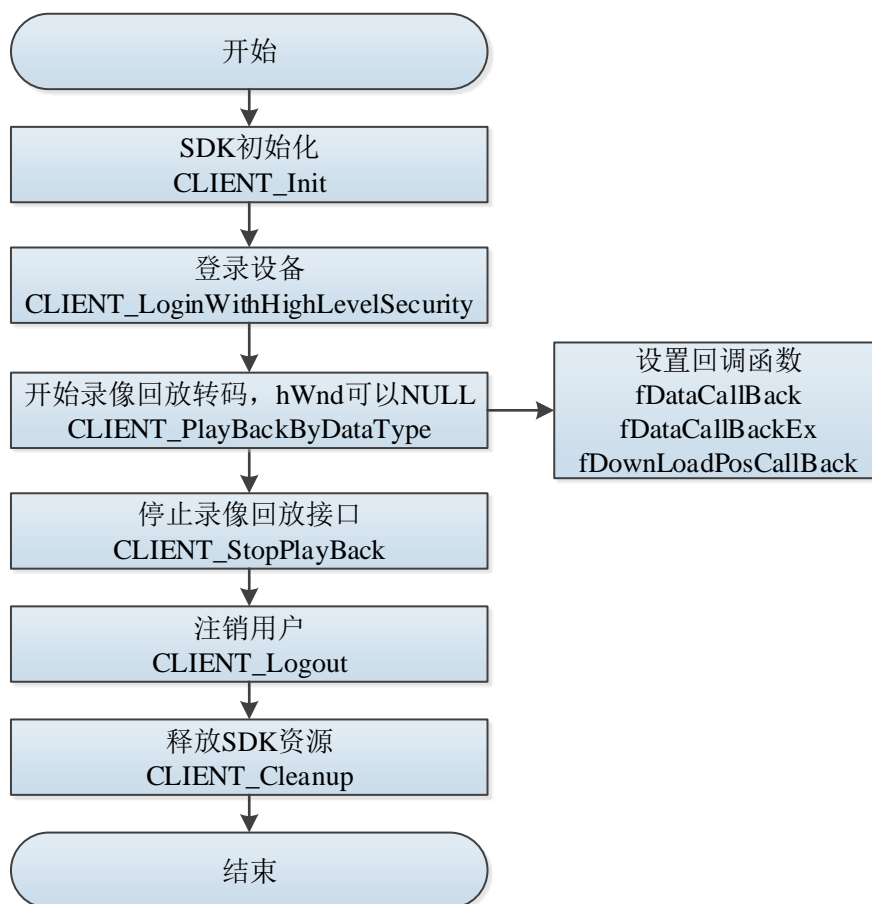
表2-13 录像回放转码的接口信息

接口	说明
CLIENT_PlayBackByDataType	开始录像回放转码接口
CLIENT_StopPlayBack	停止录像回放转码接口

2.12.3 流程说明

录像回放转码流程如图 2-16 所示。

图2-16 录像回放转码流程



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_PlayBackByDataType`，启动录像下载，参数 `hWnd` 可以为 `NULL`。
- 步骤 4 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 5 录像回放转码使用完毕后，调用 `CLIENT_StopPlayBack` 停止录像回放。
- 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。

2.12.4 示例代码

```

import com.netsdk.lib.NetSDKLib;
import com.netsdk.lib.NetSDKLib.*;
import com.sun.jna.Native;
import com.sun.jna.Pointer;
import javax.swing.*;
import java.awt.*;
    
```

```

import java.util.Vector;
public class CommonWithCallBack { // 带回调方法
    static NetSDKLib netsdkApi    = NetSDKLib.NETSDK_INSTANCE;
    LLong loginHandle;
    JWindow wnd;
    private static final int MAX_WINDOW_NUM = 4;
    Vector<JWindow> vecWnd;
    public CommonWithCallBack(LLong loginHandle)
    {
        this.loginHandle = loginHandle;
        createWindow();
    }

    public void PlayBackByDataType() {

        wnd.setVisible(true);

        NetSDKLib.NET_IN_PLAYBACK_BY_DATA_TYPE stIn = new
NetSDKLib.NET_IN_PLAYBACK_BY_DATA_TYPE();
        stIn.emDataType = EM_REAL_DATA_TYPE.EM_REAL_DATA_TYPE_GBPS; // 私
有码流
        stIn.nChannelID = 0;
        stIn.hWnd = Native.getComponentPointer(wnd);    // 播放窗格
        stIn.stStartTime.setTime(2018, 5, 22, 13, 0, 0); // 开始时间
        stIn.stStopTime.setTime(2018, 5, 22, 14, 0, 0); // 结束时间
        stIn.nPlayDirection = 0;                        // 正放
        stIn.cbDownloadPos = DownloadPosCB.getInstance();
        stIn.cbDownloadPos = PlayBackPosCallBack.getInstance();
        stIn.dwPosUser = null;

        stIn.fDownloadDataCallBack = PlayBackDataCallBack.getInstance();
        stIn.dwDataUser = null;

        NetSDKLib.NET_OUT_PLAYBACK_BY_DATA_TYPE stOut = new
NetSDKLib.NET_OUT_PLAYBACK_BY_DATA_TYPE();

        LLong IPlayHandle = netsdkApi.CLIENT_PlayBackByDataType(loginHandle, stIn,
stOut, 5000);
        if(IPlayHandle.longValue() != 0) {
            System.out.println("PlayBackByDataType Succeed!");
        } else {
            System.err.printf("PlayBackByDataType Failed!Last Error[0x%x]\n",
netsdkApi.CLIENT_GetLastError());
            return;
        }
        try {
            Thread.sleep(10000);

```

```

    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    netsdkApi.CLIENT_StopPlayBack(IPlayHandle);    // 停止回放
    wnd.setVisible(false);
}

// 回调建议写成单例模式，回调里处理数据，需要另开线程
// 回放进度回调
public static class PlayBackPosCallBack implements NetSDKLib.fDownloadPosCallBack
{
    private PlayBackPosCallBack() {}

    private static class PlayBackPosCallBackHolder {
        private static final PlayBackPosCallBack posCB = new PlayBackPosCallBack();
    }

    public static final PlayBackPosCallBack getInstance() {
        return PlayBackPosCallBackHolder.posCB;
    }

    @Override
    public void invoke(LLong IPlayHandle, int dwTotalSize, int dwDownloadSize, Pointer
dwUser) {
        System.out.println("PlayBackPosCallBack dwTotalSize:  " + dwTotalSize + "
dwDownloadSize:  " + dwDownloadSize);
    }
}

// 回放数据回调
public static class PlayBackDataCallBack implements NetSDKLib.fDataCallBack {
    private PlayBackDataCallBack() {}

    private static class PlayBackDataCallBackHolder {
        private static final PlayBackDataCallBack dataCB = new
PlayBackDataCallBack();
    }

    public static final PlayBackDataCallBack getInstance() {
        return PlayBackDataCallBackHolder.dataCB;
    }
}

```

```

@Override
public int invoke(LLong IRealHandle, int dwDataType, Pointer pBuffer, int dwBufSize,
Pointer dwUser) {

    if (dwDataType == (NetSDKLib.NET_DATA_CALL_BACK_VALUE +
EM_REAL_DATA_TYPE.EM_REAL_DATA_TYPE_GBPS)) {
        System.out.println("PlayBack DataCallBack [ " + dwDataType + " ]");
    }
    return 0;
}

// 回调建议写成单例模式，回调里处理数据，需要另开线程
// 下载进度回调
public static class DownloadPosCB implements NetSDKLib.fTimeDownLoadPosCallBack
{
    private DownloadPosCB() {}
    private static class DownloadPosCallBackHolder {
        private static final DownloadPosCB posCB = new DownloadPosCB();
    }
    public static final DownloadPosCB getInstance() {
        return DownloadPosCB.DownloadPosCallBackHolder.posCB;
    }
    @Override
    public void invoke(LLong IPlayHandle, int dwTotalSize, int dwDownLoadSize, int
index, NetSDKLib.NET_RECORDFILE_INFO.ByValue recordfileinfo, Pointer dwUser) {
        System.out.println("DownloadPosCallBack dwTotalSize:  " + dwTotalSize + "
dwDownLoadSize:  " + dwDownLoadSize);
    }
}

public void createWindow() {
    wnd = new JWindow();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    screenSize.height /= 2;
    screenSize.width /= 2;
    wnd.setSize(screenSize);

    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
    int w = wnd.getSize().width;
    int h = wnd.getSize().height;
    int x = (dim.width - w) / 2;
    int y = (dim.height - h) / 2;
    wnd.setLocation(x, y);
}
}

```


2.13 录像下载转码

2.13.1 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。

2.13.2 接口总览

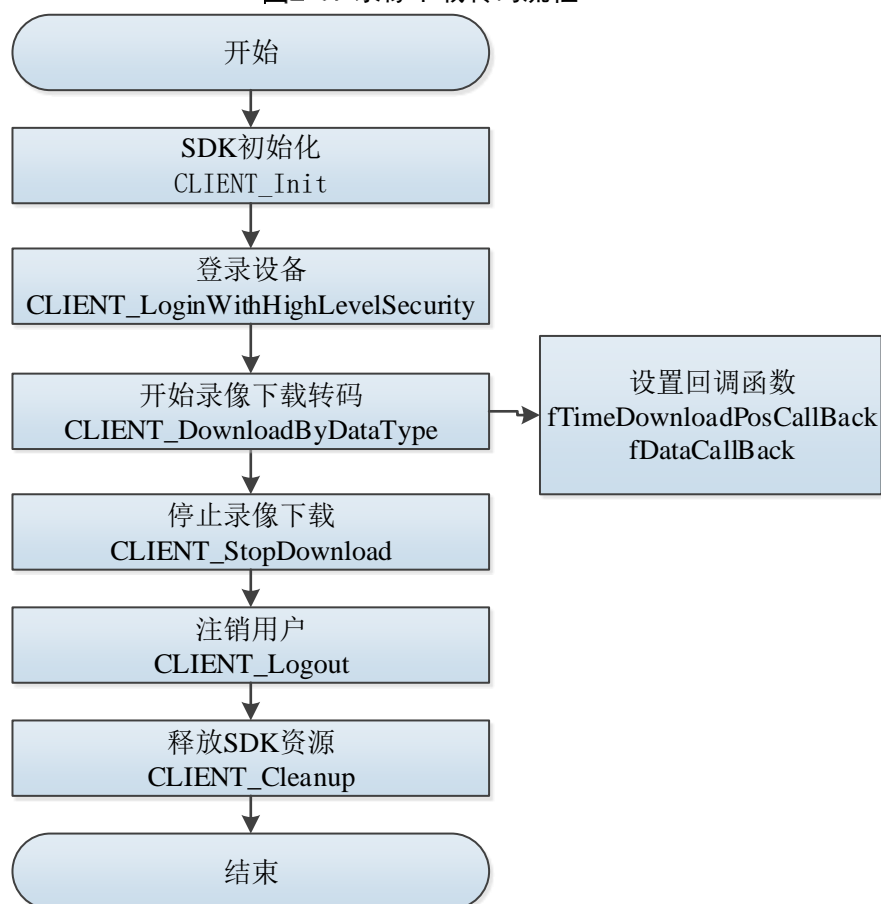
表2-14 录像回放转码的接口信息

接口	说明
CLIENT_DownloadByDataType	开始录像下载转码接口
CLIENT_StopDownload	停止录像下载转码接口

2.13.3 流程说明

录像下载转码流程如图 2-17 所示。

图2-17 录像下载转码流程



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_DownloadByDataType`，启动录像转码下载，参数 `hWnd` 可以为 `NULL`。
- 步骤 4 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 5 录像下载转码使用完毕后，调用 `CLIENT_StopDownload` 停止录像下载。
- 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 `CLIENTCleanup` 释放 SDK 资源。

2.13.4 示例代码

```

import com.netsdk.lib.NetSDKLib;
import com.netsdk.lib.NetSDKLib.*;
import com.sun.jna.Pointer;
import javax.swing.*;
import java.awt.*;
import java.util.Vector;
public class CommonWithCallBack { // 带回调方法

```

```

static NetSDKLib netsdkApi    = NetSDKLib.NETSDK_INSTANCE;
LLong loginHandle;
JWindow wnd;
private static final int MAX_WINDOW_NUM = 4;
Vector<JWindow> vecWnd;
public CommonWithCallBack(LLong loginHandle)
{
    this.loginHandle = loginHandle;
    createWindow();
}

public void DownloadByDataType() {

    NetSDKLib.NET_IN_DOWNLOAD_BY_DATA_TYPE stIn = new
NetSDKLib.NET_IN_DOWNLOAD_BY_DATA_TYPE();

    stIn.emDataType =
EM_REAL_DATA_TYPE.EM_REAL_DATA_TYPE_FLV_STREAM; // 私有码流
    stIn.emRecordType = EM_QUERY_RECORD_TYPE.EM_RECORD_TYPE_ALL; //
所有录像
    stIn.nChannelID = 0;
    stIn.stStartTime.setTime(2018, 12, 30, 12, 55, 0); // 开始时间
    stIn.stStopTime.setTime(2018, 11, 30, 13, 0, 0); // 结束时间
    stIn.cbDownLoadPos = DownloadPosCallBack.getInstance();
    stIn.dwPosUser = null;

    stIn.fDownLoadDataCallBack = DownLoadDataCallBack.getInstance();
    stIn.dwDataUser = null;
    stIn.szSavedFileName = "d:/456.dat";

    NetSDKLib.NET_OUT_DOWNLOAD_BY_DATA_TYPE stOut = new
NetSDKLib.NET_OUT_DOWNLOAD_BY_DATA_TYPE();
    stIn.write();
    stOut.write();
    LLong IPlayHandle = netsdkApi.CLIENT_DownloadByDataType(loginHandle,
stIn.getPointer(), stOut.getPointer(), 5000);
    if(IPlayHandle.longValue() != 0) {
        System.out.println("DownloadByDataType Succeed!");
    } else {
        System.err.printf("DownloadByDataType Failed!Last Error[0x%x]\n",
netsdkApi.CLIENT_GetLastError());
        return;
    }

    try {
        Thread.sleep(10000);
    } catch (InterruptedException e) {

```

```

        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    netsdkApi.CLIENT_StopDownload(IPlayHandle);    // 停止下载
}

// 回调建议写成单例模式，回调里处理数据，需要另开线程
// 下载进度回调
public static class DownloadPosCallBack implements
NetSDKLib.fTimeDownLoadPosCallBack {

    private DownloadPosCallBack() {}

    private static class DownloadPosCallBackHolder {
        private static final DownloadPosCallBack posCB = new DownloadPosCallBack();
    }

    public static final DownloadPosCallBack getInstance() {
        return DownloadPosCallBackHolder.posCB;
    }

    @Override
    public void invoke(LLong IPlayHandle, int dwTotalSize, int dwDownLoadSize, int
index, NET_RECORDFILE_INFO.ByValue recordfileinfo, Pointer dwUser) {

        System.out.println("DownloadPosCallBack dwTotalSize:  " + dwTotalSize + "
dwDownLoadSize:  " + dwDownLoadSize);
    }
}

// 下载数据回调
public static class DownLoadDataCallBack implements NetSDKLib.fDataCallBack {

    private DownLoadDataCallBack() {}

    private static class DownloadDataCallBackHolder {
        private static final DownLoadDataCallBack dataCB = new
DownLoadDataCallBack();
    }

    public static final DownLoadDataCallBack getInstance() {
        return DownloadDataCallBackHolder.dataCB;
    }

    @Override
    public int invoke(LLong IRealHandle, int dwDataType, Pointer pBuffer, int dwBufSize,

```

```
Pointer dwUser) {
    if (dwDataType == (NetSDKLib.NET_DATA_CALL_BACK_VALUE +
EM_REAL_DATA_TYPE.EM_REAL_DATA_TYPE_GBPS)) {
        System.out.println("DownLoad DataCallBack [ " + dwDataType + " ]");
    }
    return 0;
}
}

public void createWindow() {
    wnd = new JWindow();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    screenSize.height /= 2;
    screenSize.width /= 2;
    wnd.setSize(screenSize);

    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
    int w = wnd.getSize().width;
    int h = wnd.getSize().height;
    int x = (dim.width - w) / 2;
    int y = (dim.height - h) / 2;
    wnd.setLocation(x, y);
}
}
```

2.14 远程通道

2.14.1 简介

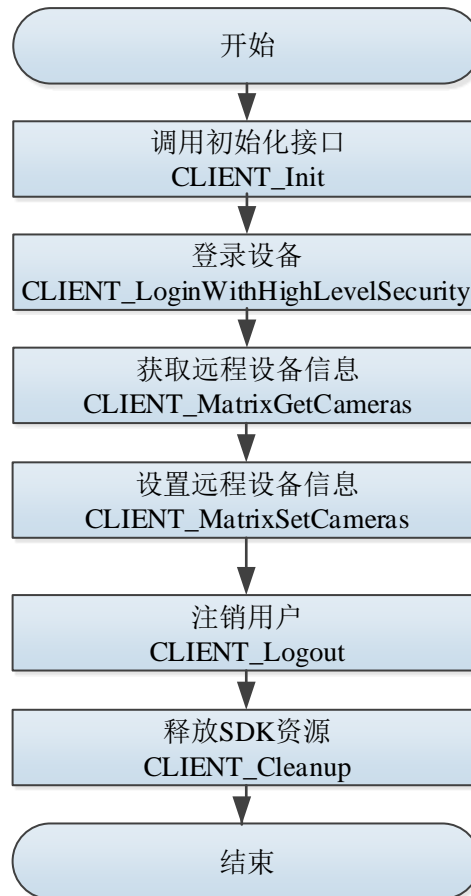
包含获取与配置远程设备信息（如设备型号、IP 等消息）的接口

2.14.2 接口总览

表2-15 远程通道的接口信息

接口	说明
CLIENT_MatrixGetCameras	获取远程设备信息，如设备型号、IP 等信息
CLIENT_MatrixSetCameras	设置远程设备信息

2.14.3 流程说明



流程说明

- 步骤 1 调用 CLIENT_Init 完成 SDK 初始化流程。
- 步骤 2 初始化成功后，调用 CLIENT_LoginWithHighLevelSecurity 登录设备。
- 步骤 3 调用 CLIENT_MatrixGetCameras 获取远程设备信息，如设备型号、IP 等信息。
- 步骤 4 调用 CLIENT_MatrixSetCameras 设置远程设备信息，如设备型号、IP 等信息。
- 步骤 5 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。

2.14.4 示例代码

```
public void MatrixGetCameras () throws UnsupportedOperationException {  
  
    // 显示源信息数组初始化  
    int nCameraCount = 60;  
    int nMaxVideoInputCount = 10; //视频输入通道最大数  
    NET_MATRIX_CAMERA_INFO[] cameras = new  
NET_MATRIX_CAMERA_INFO[nCameraCount];  
    // 视频输入通道信息数组初始化  
    NetSDKLib.NET_VIDEO_INPUTS[] inputs= new
```

```

NetSDKLib.NET_VIDEO_INPUTS[nMaxVideoInputCount];
    for (int j = 0; j < nMaxVideoInputCount; j++) {
        inputs[j] = new NetSDKLib.NET_VIDEO_INPUTS();
    }
    for(int i = 0; i < nCameraCount; i++) {
        NetSDKLib.NET_MATRIX_CAMERA_INFO camera = new
NetSDKLib.NET_MATRIX_CAMERA_INFO();
        NetSDKLib.NET_REMOTE_DEVICE device = new
NetSDKLib.NET_REMOTE_DEVICE();
        device.nMaxVideoInputCount = nMaxVideoInputCount; // 视频输入通道最大数

        device.pstuVideoInputs = new Memory(inputs[0].size() * nMaxVideoInputCount);

        device.pstuVideoInputs.clear(inputs[0].size() * nMaxVideoInputCount);
        // 将数组内存拷贝到 Pointer
        ToolKits.SetStructArrToPointerData(inputs, device.pstuVideoInputs);
        camera.stuRemoteDevice = device;
        cameras[i] = camera;
    }

    /*
    * 入参
    */
    NetSDKLib.NET_IN_MATRIX_GET_CAMERAS stuIn = new
NetSDKLib.NET_IN_MATRIX_GET_CAMERAS();

    /*
    * 出参
    */
    NET_OUT_MATRIX_GET_CAMERAS stuOut = new
NET_OUT_MATRIX_GET_CAMERAS();
    stuOut.nMaxCameraCount = nCameraCount;
    stuOut.pstuCameras = new Memory(cameras[0].size() * nCameraCount);
    stuOut.pstuCameras.clear(cameras[0].size() * nCameraCount);

    ToolKits.SetStructArrToPointerData(cameras, stuOut.pstuCameras); // 将数组内存
拷贝到 Pointer

    if(netsdkApi.CLIENT_MatrixGetCameras(loginHandle, stuIn, stuOut, 5000)) {
        ToolKits.GetPointerDataToStructArr(stuOut.pstuCameras, cameras); // 将
Pointer 的内容 输出到 数组

```

```

        for(int j = 0; j < stuOut.nRetCameraCount; j++) {
            if(cameras[j].bRemoteDevice == 0) {
                System.out.println("NVR 通道号: " + cameras[j].nUniqueChannel);
                System.out.println("前端通道号(远程设备): " + cameras[j].nChannelID);
                System.out.println("设备 ID : " + new String(cameras[j].szDevID).trim());
                //System.out.println("IP : " + new
String(cameras[j].stuRemoteDevice.szIp).trim());
                //System.out.println("nPort : " + cameras[j].stuRemoteDevice.nPort);
                //System.out.println("szUser : " + new
String(cameras[j].stuRemoteDevice.szUser).trim());
                //System.out.println("szPwd : " + new
String(cameras[j].stuRemoteDevice.szPwd).trim());
                System.out.println("通道个数 : " +
cameras[j].stuRemoteDevice.nVideoInputChannels + "\n");
            } else {
                System.out.println("NVR 通道号: " + cameras[j].nUniqueChannel);
                System.out.println("前端通道号(远程设备): " + cameras[j].nChannelID);
                System.out.println("设备 ID : " + new String(cameras[j].szDevID).trim());
                System.out.println("设备类型 : " + new
String(cameras[j].stuRemoteDevice.szDevClass).trim());
                System.out.println("设备序列号 : " + new
String(cameras[j].stuRemoteDevice.szSerialNo).trim());
                System.out.println("实际返回通道个数 : " +
cameras[j].stuRemoteDevice.nRetVideoInputCount);

                ToolKits.GetPointerDataToStructArr(cameras[j].stuRemoteDevice.pstuVideoInputs,
inputs); // 将 Pointer 的内容 输出到 数组
                for (int i = 0; i < cameras[j].stuRemoteDevice.nRetVideoInputCount; i++)
                {
                    System.out.println("使能: "+inputs[i].bEnable);
                    System.out.println("通道名称: "+new
String(inputs[i].szChnName,"GBK"));
                    System.out.println("控制 ID: "+new String(inputs[i].szControlID));
                    System.out.println("主码流 url 地址: "+new
String(inputs[i].szMainStreamUrl)+"\n");
                }
            }
        }
    } else {
        System.err.println("获取所有有效显示源失败! " + ToolKits.getErrorCode());
    }
}

```



```

    }

    /**
     * 设置显示源
     */
    public void MatrixSetCameras() {
        // 设置的显示源个数
        int nCameraCount = 1;

        // 显示源信息数组初始化
        NET_MATRIX_CAMERA_INFO[] cameras = new
NET_MATRIX_CAMERA_INFO[nCameraCount];
        for(int i = 0; i < nCameraCount; i++) {
            cameras[i] = new NET_MATRIX_CAMERA_INFO();
        }

        cameras[0].nUniqueChannel = 0;
        cameras[0].nChannelID = 0; // 通道号

        /**
         * 入参
         */
        NET_IN_MATRIX_SET_CAMERAS stuIn = new
NET_IN_MATRIX_SET_CAMERAS();
        stuIn.nCameraCount = nCameraCount;
        stuIn.pstuCameras = new Memory(cameras[0].size() * nCameraCount);
        stuIn.pstuCameras.clear(cameras[0].size() * nCameraCount);

        ToolKits.SetStructArrToPointerData(cameras, stuIn.pstuCameras); // 将数组内存拷
        贝到 Pointer

        /**
         * 出参
         */
        NET_OUT_MATRIX_SET_CAMERAS stuOut = new
NET_OUT_MATRIX_SET_CAMERAS();

        if(netsdkApi.CLIENT_MatrixSetCameras(loginHandle, stuIn, stuOut, 5000)) {
            System.out.println("设置显示源成功！");
        } else {

```

```
        System.err.println("设置显示源失败！" + ToolKits.getErrorCode());
    }
}
```

第 3 章 接口函数

3.1 SDK 初始化

3.1.1 SDK 初始化 CLIENT_Init

表3-1 SDK 初始化 CLIENT_Init

选项	说明	
描述	对整个 SDK 进行初始化	
方法	public boolean CLIENT_Init(Callback cbDisconnect, Pointer dwUser);	
参数	[in]cbDisconnect	断线回调函数
	[in]dwUser	断线回调函数的用户参数
返回值	成功返回 TRUE，失败返回 FALSE	
说明	<ul style="list-style-type: none">调用网络 SDK 其他函数的前提回调函数设置成 NULL 时，设备断线后不会回调给用户	

3.1.2 SDK 清理 CLIENT_Cleanup

表3-2 SDK 清理 CLIENT_Cleanup

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

3.1.3 设置断线重连回调函数 CLIENT_SetAutoReconnect

表3-3 设置断线重连回调函数 CLIENT_SetAutoReconnect

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

3.1.4 设置网络参数 CLIENT_SetNetworkParam

表3-4 设置网络参数 CLIENT_SetNetworkParam

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

3.2 设备登录

3.2.1 用户登录设备 CLIENT_LoginWithHighLevelSecurity

表3-5 用户登录设备 CLIENT_LoginWithHighLevelSecurity

选项	说明	
描述	用户登录设备	
方法	public LLong CLIENT_LoginWithHighLevelSecurity(NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY pstInParam, NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY pstOutParam);	
参数	[in]pstInParam	输入参数
	[out]pstOutParam	输出参数
返回值	成功返回登录句柄，失败返回 0	
说明	此方法封装在 NetSDKLib 接口中，通常通过以下方式调用： m_hLoginHandle = netsdk.CLIENT_CLIENT_LoginWithHighLevelSecurity(pstInParam, pstOutParam);	

3.2.2 用户登出设备 CLIENT_Logout

表3-6 用户登出设备 CLIENT_Logout

选项	说明	
描述	用户登出设备	
方法	public boolean CLIENT_Logout(LLong lLoginID);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	此方法封装在 NetSDKLib 接口中，通常通过以下方式调用 netsdk.CLIENT_Logout(m_hLoginHandle);	

3.3 实时预览

3.3.1 打开预览 CLIENT_RealPlayEx

表3-7 打开预览 CLIENT_RealPlayEx

选项	说明	
描述	打开实时预览	
方法	public LLong CLIENT_RealPlayEx(LLong ILoginID, int nChannelID, Pointer hWnd, int rType);	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[in]nChannelID	视频通道号，从 0 开始递增的整数
	[in]hWnd	窗口句柄，仅在 Windows 系统下有效
	[in]rType	预览类型
返回值	成功返回非 0，失败返回 0	
说明	在 Windows 环境下： <ul style="list-style-type: none">hWnd 为有效值时，在对应窗口显示画面hWnd 为 NULL 时，表示取流方式，通过设置回调函数来获取视频数据，交由用户处理	

预览类型及含义请参见表 3-8。

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

3.3.2 关闭预览 CLIENT_StopRealPlayEx

表3-9 关闭预览 CLIENT_StopRealPlayEx

选项	说明	
描述	关闭实时预览	
方法	public boolean CLIENT_StopRealPlayEx(LLong IRealHandle);	
参数	[in]IRealHandle	CLIENT_RealPlayEx 的返回值

选项	说明
返回值	成功返回 TRUE，失败返回 FALSE
说明	无

3.3.3 保存预览数据 CLIENT_SaveRealData

表3-10 保存预览数据 CLIENT_SaveRealData

选项	说明	
描述	保存实时预览数据为文件	
方法	public boolean CLIENT_SaveRealData(LLong IRealHandle, String pchFileName);	
参数	[in]IRealHandle	CLIENT_RealPlayEx 的返回值
	[in]pchFileName	需要保存的文件路径
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.3.4 停止保存预览数据 CLIENT_StopSaveRealData

表3-11 停止保存预览数据 CLIENT_StopSaveRealData

选项	说明	
描述	停止保存实时预览数据为文件	
方法	public boolean CLIENT_StopSaveRealData(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	

3.3.5 设置预览数据回调 CLIENT_SetRealDataCallBackEx

表3-12 设置预览数据回调 CLIENT_SetRealDataCallBackEx

选项	说明	
描述	设置实时预览数据回调	
方法	public boolean CLIENT_SetRealDataCallBackEx(LLong IRealHandle, StdCallCallback cbRealData, Pointer dwUser, int dwFlag);	
参数	[in]IRealHandle	CLIENT_RealPlayEx 的返回值
	[in]cbRealData	预览数据流回调函数
	[in]dwUser	预览数据流回调函数的参数
	[in]dwFlag	回调中预览数据的类型
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

回调中预览数据的类型请参见表 3-13。

表3-13 dwFlag 类型及含义

dwFlag	含义
0x00000001	设备的原始数据

dwFlag	含义
0x00000004	转成 YUV 格式的数据

3.4 抓图

3.4.1 同步抓图 CLIENT_SnapPictureToFile

表3-14 同步抓图 CLIENT_SnapPictureToFile

选项	说明	
描述	同步抓图	
方法	public boolean CLIENT_SnapPictureToFile(LLong lLoginID, Pointer pInParam, Pointer pOutParam, int nWaitTime)	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[in]pInParam	输入参数, 参考 NET_IN_SNAP_PIC_TO_FILE_PARAM
	[out]pOutParam	输出参数, 参考 NET_OUT_SNAP_PIC_TO_FILE_PARAM
	[in]nWaitTime	超时时间, 单位: 毫秒
返回值	成功返回 TRUE, 失败返回 FALSE	
说明	<ul style="list-style-type: none"> 同步接口, 由设备抓取图片, 通过网络传给用户 需要设备支持该功能 	

3.4.2 本地抓图 CLIENT_CapturePictureEx

表3-15 本地抓图 CLIENT_CapturePictureEx

选项	说明	
描述	本地抓图	
方法	public boolean CLIENT_CapturePictureEx(LLong hPlayHandle, String pchPicFileName, int eFormat);	
参数	[in]hPlayHandle	CLIENT_RealPlayEx 的返回值
	[in]pchPicFileName	需要保存的文件路径
	[in]eFormat	图片格式
返回值	成功返回 TRUE, 失败返回 FALSE	
说明	<ul style="list-style-type: none"> 同步接口, 将图片数据直接写成文件 图片从设备发送过来的实时预览数据流中抓取 	

3.4.3 异步抓图 CLIENT_SnapPictureEx

表3-16 异步抓图 CLIENT_SnapPictureEx

选项	说明	
描述	异步抓图	
方法	public boolean CLIENT_SnapPictureEx(LLong ILoginID, SNAP_PARAMS stParam, IntByReference reserved);	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[in]stParam	抓图参数结构体
	[in]reserved	图片格式
返回值	成功返回 TRUE，失败返回 FALSE	
说明	<ul style="list-style-type: none">同步接口，将图片数据直接写成文件图片从设备发送过来的实时预览数据流中抓取	

3.4.4 设置异步抓图回调

表3-17 设置异步抓图回调

选项	说明	
描述	异步抓图回调函数	
方法	public void CLIENT_SetSnapRevCallBack(Callback OnSnapRevMessage, Pointer dwUser);	
参数	[out]OnSnapRevMessage	抓图回调函数原形
	[out]dwUser	回调函数的用户参数
返回值	空	
说明	无	

3.5 云台控制

3.5.1 云台控制 CLIENT_DHPTZControlEx2

表3-18 云台控制 CLIENT_DHPTZControlEx2

选项	说明	
描述	云台控制 CLIENT_DHPTZControlEx	
方法	public boolean CLIENT_DHPTZControlEx2(LLong ILoginID, int nChannelID, int dwPTZCommand, int IParam1, int IParam2, int IParam3, int dwStop,Pointer param4);	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[in]nChannelID	视频通道号，从 0 开始递增的整数
	[in]dwPTZCommand	控制命令类型
	[in]IParam1	参数 1
	[in]IParam2	参数 2

选项	说明	
	[in]Param3	参数 3
	[in]dwStop	停止标志。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
	[in]param4	支持扩展控制命令参数，主要支持如下几种控制命令： NET_EXTPTZ_MOVE_ABSOLUTELY NET_EXTPTZ_MOVE_CONTINUOUSLY NET_EXTPTZ_GOTOPRESET NET_EXTPTZ_SET_VIEW_RANGE NET_EXTPTZ_FOCUS_ABSOLUTELY NET_EXTPTZ_HORSECTORSCAN NET_EXTPTZ_VERSECTORSCAN NET_EXTPTZ_SET_FISHEYE_EPTZ NET_EXTPTZ_AUXIOPEN NET_EXTPTZ_AUXICLOSE NET_EXTPTZ_SET_TRACK_START NET_EXTPTZ_SET_TRACK_STOP NET_EXTPTZ_INTELLI_TRACKMOVE NET_EXTPTZ_SET_FOCUS_REGION NET_EXTPTZ_INTELLI_SETLENSWISDOMSTATE NET_EXTPTZ_INTELLI_SETFOCUSAREA NET_EXTPTZ_SINGLEDIRECTIONCALIBRATION NET_EXTPTZ_MOVE_RELATIVELY NET_EXTPTZ_SET_DIRECTION NET_EXTPTZ_BASE_MOVE_ABSOLUTELY NET_EXTPTZ_BASE_MOVE_CONTINUOUSLY NET_EXTPTZ_BASE_SET_FOCUS_MAP_VALUE NET_EXTPTZ_BASE_MOVE_ABSOLUTELY_ONLYPT NET_EXTPTZ_BASE_MOVE_ABSOLUTELY_ONLYZOOM NET_EXTPTZ_STOP_MOVE NET_EXTPTZ_START NET_EXTPTZ_STOP NET_EXTPTZ_START_PATTERN_RECORD NET_EXTPTZ_STOP_PATTERN_RECORD NET_EXTPTZ_START_PATTERN_REPLAY NET_EXTPTZ_STOP_PATTERN_REPLAY NET_EXTPTZ_MOVE_DIRECTLY
返回值	成功返回 TRUE，失败返回 FALSE	
说明	dwPTZCommand 与 Param1、Param2 和 Param3 的关系，请参见表 3-19。	

dwPTZCommand 与 Param1、Param2 和 Param3 的关系，请参见表 3-19。

表3-19 Param1、Param2 和 Param3 的关系

dwPTZCommand 宏定义	功能描述	param1	param2	param3
NET_PTZ_UP_CONTROL	上	-	垂直速度（1-8）	-
NET_PTZ_DOWN_CONTROL	下	-	垂直速度（1-8）	-

dwPTZCommand 宏定义	功能描述	param1	param2	param3
NET_PTZ_LEFT_CONTROL	左	-	水平速度（1-8）	-
NET_PTZ_RIGHT_CONTROL	右	-	水平速度（1-8）	-
NET_PTZ_ZOOM_ADD_CONTROL	变倍+	-	倍速	-
NET_PTZ_ZOOM_DEC_CONTROL	变倍-	-	倍速	-
NET_PTZ_FOCUS_ADD_CONTROL	调焦+	-	倍速	-
NET_PTZ_FOCUS_DEC_CONTROL	调焦-	-	倍速	-
NET_PTZ_APERTURE_ADD_CONTROL	光圈+	-	倍速	-
NET_PTZ_APERTURE_DEC_CONTROL	光圈-	-	倍速	-
NET_PTZ_POINT_MOVE_CONTROL	转至预置点	-	预置点值	-
NET_PTZ_POINT_SET_CONTROL	设置	-	预置点值	-
NET_PTZ_POINT_DEL_CONTROL	删除	-	预置点值	-
NET_PTZ_POINT_LOOP_CONTROL	点间轮循	巡航线路	-	76: 开始 99: 自动 96: 停止
NET_PTZ_LAMP_CONTROL	灯光雨刷	0x01: 开启 x00: 关闭	-	-
NET_EXTPTZ_LEFTTOP	左上	垂直速度（1-8）	水平速度（1-8）	-
NET_EXTPTZ_RIGHTTOP	右上	垂直速度（1-8）	水平速度（1-8）	-
NET_EXTPTZ_LEFTDOWN	左下	垂直速度（1-8）	水平速度（1-8）	-
NET_EXTPTZ_RIGHTDOWN	右下	垂直速度（1-8）	水平速度（1-8）	-
NET_EXTPTZ_ADDTOLOOP	加入预置点到巡航	巡航线路	预置点值	-
NET_EXTPTZ_DELFROMLOOP	删除巡航中预置点	巡航线路	预置点值	-
NET_EXTPTZ_CLOSELOOP	清除巡航	巡航线路	-	-
NET_EXTPTZ_STARTPANCUISE	开始水平旋转	-	-	-
NET_EXTPTZ_STOPPANCUISE	停止水平旋转	-	-	-
NET_EXTPTZ_SETLEFTBORDER	设置左边界	-	-	-

dwPTZCommand 宏定义	功能描述	param1	param2	param3
NET_EXTPTZ_RIGHTBORDER	设置右边界	-	-	-
NET_EXTPTZ_STARTLINESCAN	开始线扫	-	-	-
NET_EXTPTZ_CLOSELINESCAN	停止线扫	-	-	-
NET_EXTPTZ_SETMODESTART	设置模式开始	模式线路	-	-
NET_EXTPTZ_SETMODESTOP	设置模式结束	模式线路	-	-
NET_EXTPTZ_RUNMODE	运行模式	模式线路	-	-
NET_EXTPTZ_STOPMODE	停止模式	模式线路	-	-
NET_EXTPTZ_DELETEMODE	清除模式	模式线路	-	-
NET_EXTPTZ_REVERSECOMM	翻转命令	-	-	-
NET_EXTPTZ_FASTGOTO	快速定位	水平坐标 (0-8192)	垂直坐标 (0-8192)	变倍 (4)
NET_EXTPTZ_AUXIOPEN	辅助开关开	辅助点	-	-
NET_EXTPTZ_AUXICLOSE	辅助开关关	辅助点	-	-
NET_EXTPTZ_OPENMENU	打开球机菜单	-	-	-
NET_EXTPTZ_CLOSEMENU	关闭菜单	-	-	-
NET_EXTPTZ_MENUOK	菜单确定	-	-	-
NET_EXTPTZ_MENUCANCEL	菜单取消	-	-	-
NET_EXTPTZ_MENUUP	菜单上	-	-	-
NET_EXTPTZ_MENUDOWN	菜单下	-	-	-
NET_EXTPTZ_MENULEFT	菜单左	-	-	-
NET_EXTPTZ_MENURIGHT	菜单右	-	-	-
NET_EXTPTZ_ALARMHANDLE	报警联动云台	报警输入通道	报警联动类型： ● 预置点 ● 线扫 ● 巡航	联动值，如预置点号
NET_EXTPTZ_MATRIXSWITCH	矩阵切换	预览器号 (视频输出号)	视频输入号	矩阵号
NET_EXTPTZ_LIGHTCONTROL	灯光控制器	参考 DH_PTZ_LAMP_CONTROL	-	-
NET_EXTPTZ_EXACTGOTO	三维精确定位	水平角度 (0 ~ 3600)	垂直坐标 (0 ~ 900)	变倍 (1 ~ 128)
NET_EXTPTZ_RESETZERO	三维定位重设零位	-	-	-
NET_EXTPTZ_UP_TELE	上+TELE	速度 (1-8)	-	-
NET_EXTPTZ_DOWN_TELE	下+TELE	速度 (1-8)	-	-
NET_EXTPTZ_LEFT_TELE	左+TELE	速度 (1-8)	-	-

dwPTZCommand 宏定义	功能描述	param1	param2	param3
NET_EXTPTZ_RIGHT_TELE	右+TELE	速度（1-8）	-	-
NET_EXTPTZ_LEFTUP_TELE	左上+TELE	速度（1-8）	-	-
NET_EXTPTZ_LEFTDOWN_TELE	左下+TELE	速度（1-8）	-	-
NET_EXTPTZ_TIGHTUP_TELE	右上+TELE	速度（1-8）	-	-
NET_EXTPTZ_RIGHTDOWN_TELE	右下+TELE	速度（1-8）	-	-
NET_EXTPTZ_UP_WIDE	上+WIDE	速度（1-8）	-	-
NET_EXTPTZ_DOWN_WIDE	下+WIDE	速度（1-8）	-	-
NET_EXTPTZ_LEFT_WIDE	左+WIDE	速度（1-8）	-	-
NET_EXTPTZ_RIGHT_WIDE	右+WIDE	速度（1-8）	-	-
NET_EXTPTZ_LEFTUP_WIDE	左上+WIDE	速度（1-8）	-	-
NET_EXTPTZ_LEFTDOWN_WIDE	左下+WIDE	速度（1-8）	-	-
NET_EXTPTZ_RIGHTUP_WIDE	右上+WIDE	速度（1-8）	-	-
NET_EXTPTZ_RIGHTDOWN_WIDE	右下+WIDE	速度（1-8）	-	-

3.6 语音对讲

3.6.1 开启对讲 CLIENT_StartTalkEx

表3-20 开启对讲 CLIENT_StartTalkEx

选项	说明	
描述	打开语音对讲	
方法	public LLong CLIENT_StartTalkEx(LLong ILoginID, Callback pfcb, Pointer dwUser);	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[in]pfcb	音频数据回调函数
	[in]dwUser	音频数据回调函数的参数
返回值	成功返回非 0，失败返回 0	
说明	无	

3.6.2 关闭对讲 CLIENT_StopTalkEx

表3-21 关闭对讲 CLIENT_StopTalkEx

选项	说明	
描述	关闭语音对讲	
方法	public boolean CLIENT_StopTalkEx(LLong ITalkHandle);	
参数	[in]ITalkHandle	CLIENT_StartTalkEx 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.6.3 发送语音 CLIENT_TalkSendData

表3-22 发送语音 CLIENT_TalkSendData

选项	说明	
描述	发送音频数据给设备	
方法	public LLong CLIENT_TalkSendData(LLong ITalkHandle, Pointer pSendBuf, int dwBufSize);	
参数	[in]ITalkHandle	CLIENT_StartTalkEx 的返回值
	[in]pSendBuf	需要发送的音频数据块的指针
	[in]dwBufSize	需要发送的音频数据块的长度，单位：字节
返回值	成功返回音频数据块的长度，失败返回-1	
说明	无	

3.6.4 解码语音 CLIENT_AudioDecEx

表3-23 解码语音 CLIENT_AudioDecEx

选项	说明	
描述	解码音频数据	
方法	public boolean CLIENT_AudioDecEx(LLong ITalkHandle, Pointer pAudioDataBuf, int dwBufSize);	
参数	[in]ITalkHandle	CLIENT_StartTalkEx 的返回值
	[in]pAudioDataBuf	需要解码的音频数据块的指针
	[in]dwBufSize	需要解码的音频数据块的长度，单位：字节
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.7 报警监听

3.7.1 开始报警监听 CLIENT_StartListenEx

表3-24 开始报警监听 CLIENT_StartListenEx

选项	说明	
描述	开启报警监听	
方法	public boolean CLIENT_StartListenEx(LLong ILoginID);	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
返回值	成功返回非 0，失败返回 0	
说明	无	

3.7.2 停止报警监听 CLIENT_StopListen

表3-25 停止报警监听 CLIENT_StopListen

选项	说明	
描述	停止报警监听	
方法	public boolean CLIENT_StopListen(LLong ILoginID);	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.7.3 设置报警回调 CLIENT_SetDVRMessCallBack

表3-26 设置报警回调 CLIENT_SetDVRMessCallBack

选项	说明	
描述	设置报警监听	
方法	public void CLIENT_SetDVRMessCallBack(Callback cbMessage , Pointer dwUser);	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[in]dwUser	返回的用户信息
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.8 智能订阅

3.8.1 开始智能事件订阅 CLIENT_RealLoadPictureEx

表3-27 开始智能事件订阅 CLIENT_RealLoadPictureEx

选项	说明	
描述	开始智能事件订阅	
方法	public LLong CLIENT_RealLoadPictureEx(LLong ILoginID, int nChannelID, int dwAlarmType, int bNeedPicFile, StdCallCallback cbAnalyzerData, Pointer dwUser, Pointer Reserved);	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[in]nChannelID	设备通道号(从 0 开始)
	[in]dwAlarmType	订阅报警事件类型
	[in]bNeedPicFile	是否订阅图片文件
	[in]cbAnalyzerData	智能事件回调函数
	[in]dwUser	用户自定义数据类型
	[in]Reserved	保留字段
返回值	成功返回 LLONG 类型订阅句柄，失败返回 0	

选项	说明
说明	接口返回失败，请使用 <code>CLIENT_GetLastError</code> 获取错误码

智能报警事件类型说明请参见表 3-28。

表3-28 智能事件类型说明

dwAlarmType 宏定义	宏定义值	含义	回调 pAlarmInfo 对应结构体
EVENT_IVS_ALL	0x00000001	所有事件	无
EVENT_IVS_CROSSFENCEDETECTION	0x0000011F	穿越围栏	DEV_EVENT_CROSSFENCEDETECTION_INFO
EVENT_IVS_CROSSLINEDETECTION	0x00000002	绊线入侵	DEV_EVENT_CROSSLINE_INFO
EVENT_IVS_CROSSREGIONDETECTION	0x00000003	区域入侵	DEV_EVENT_CROSSREGION_INFO
EVENT_IVS_LEFTDETECTION	0x00000005	物品遗留	DEV_EVENT_LEFT_INFO
EVENT_IVS_PRESERVATION	0x00000008	物品保全	DEV_EVENT_PRESERVATION_INFO
EVENT_IVS_TAKENAWAYDETECTION	0x00000115	物品搬移	DEV_EVENT_TAKENAWAYDETECTION_INFO
EVENT_IVS_WANDERDETECTION	0x00000007	徘徊事件	DEV_EVENT_WANDER_INFO
EVENT_IVS_VIDEOABNORMALDETECTION	0x00000013	视频异常	DEV_EVENT_VIDEOABNORMALDETECTION_INFO
EVENT_IVS_AUDIO_ABNORMALDETECTION	0x00000126	声音异常	DEV_EVENT_IVS_AUDIO_ABNORMALDETECTION_INFO
EVENT_IVS_CLIMBDETECTION	0x00000128	攀高检测	DEV_EVENT_IVS_CLIMB_INFO
EVENT_IVS_FIGHTDETECTION	0x0000000E	斗殴检测	DEV_EVENT_FLOWSTAT_INFO
EVENT_IVS_LEAVEDETECTION	0x00000129	离岗检测	DEV_EVENT_IVS_LEAVE_INFO
EVENT_IVS_PSRISEDETECTION	0x0000011E	起身检测	DEV_EVENT_PSRISEDETECTION_INFO
EVENT_IVS_PASTEDETECTION	0x00000004	非法黏贴物贴条检测	DEV_EVENT_PASTE_INFO

3.8.2 停止智能事件订阅 CLIENT_StopLoadPic

表3-29 停止智能事件订阅 CLIENT_StopLoadPic

选项	说明	
描述	停止智能事件订阅	
方法	public boolean CLIENT_StopLoadPic(LLong lAnalyzerHandle);	
参数	[in]lAnalyzerHandle	智能事件订阅句柄

选项	说明
返回值	BOOL 类型 <ul style="list-style-type: none"> 成功: TRUE 失败: FALSE
说明	接口返回失败, 请使用 CLIENT_GetLastError 获取错误码

3.9 录像回放

3.9.1 按时间方式回放 CLIENT_PlayBackByTimeEx

表3-30 按时间方式回放 CLIENT_PlayBackByTimeEx

选项	说明	
描述	按时间方式回放	
方法	<pre>public LLong CLIENT_PlayBackByTimeEx(LLong ILoginID, int nChannelID, NET_TIME IpStartTime, NET_TIME IpStopTime, Pointer hWnd, Callback cbDownloadPos, Pointer dwPosUser, Callback fDownloadDataCallBack, Pointer dwDataUser);</pre>	
参数	[in]ILoginID	登录句柄
	[in]nChannelID	设备通道号 (从 0 开始)
	[in]IpStartTime	开始时间
	[in]IpStopTime	结束时间
	[in]hWnd	窗口句柄, 仅在 Windows 系统下有效
	[in]cbDownloadPos	fDownloadPosCallBack 回调
	[out]dwPosUser	
	[out]fDownloadDataCallBack	fDataCallBack 回调
	[in]dwDataUser	
返回值	成功返回网络回放 ID, 失败返回 0	
说明	无	

3.9.2 设置工作模式 CLIENT_SetDeviceMode

表3-31 设置工作模式 CLIENT_SetDeviceMode

选项	说明	
描述	设置工作模式	
方法	<pre>public boolean CLIENT_SetDeviceMode(LLong ILoginID, int emType, Pointer pValue);</pre>	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 返回值
	[in]emType	工作模式枚举
	[in]pValue	相应工作模式对应的结构体
返回值	成功返回 TRUE , 失败返回 FALSE	
说明	无	

表3-32 工作模式枚举及结构体对照表

emType 枚举	含义	结构体
DH_RECORD_STREAM_TYPE	设置待查询及按时间回放的录像码流类型 <ul style="list-style-type: none"> ● 0: 主辅码流 ● 1: 主码流 ● 2: 辅码流 	无
DH_RECORD_TYPE	设置按时间录像回放及下载的录像文件类型	NET_RECORD_TYPE

3.9.3 停止录像回放 CLIENT_StopPlayBack

表3-33 停止录像回放 CLIENT_StopPlayBack

选项	说明	
描述	停止录像回放	
方法	public boolean CLIENT_StopPlayBack(LLong IPlayHandle);	
参数	[in]IPlayHandle	回放接口返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.9.4 暂停或恢复录像回放 CLIENT_PausePlayBack

表3-34 暂停或恢复录像回放 CLIENT_PausePlayBack

选项	说明	
描述	暂停或恢复录像回放	
方法	public boolean CLIENT_PausePlayBack(LLong IPlayHandle, int bPause);	
参数	[in]IPlayHandle	回放接口返回值
	[out]bPause	网络回放暂停与恢复播放参数 <ul style="list-style-type: none"> ● 1: 暂停 ● 0: 恢复
返回值	成功返回 TRUE，失败返回 FALSE	
说明	对已经打开的播放进行暂停和恢复控制	

3.10 录像下载

3.10.1 查询时间段内的所有录像文件 CLIENT_QueryRecordFile

表3-35 查询时间段内的所有录像文件 CLIENT_QueryRecordFile

选项	说明
描述	查询时间段内的所有录像文件

选项	说明	
方法	<pre>public boolean CLIENT_QueryRecordFile(LLong lLoginID, int nChannelId, int nRecordFileType, NET_TIME tmStart, NET_TIME tmEnd, String pchCardid, NET_RECORDFILE_INFO[] stFileInfo, int maxlen, IntByReference filecount, int waittime, boolean bTime);</pre>	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 返回值
	[in]nChannelId	设备通道号，从 0 开始
	[in]nRecordFileType	录像文件类型
	[in]tmStart	录像开始时间
	[in]tmEnd	录像结束时间
	[in]pchCardid	卡号
	[out]nriFileInfo	返回的录像文件信息，是一个 LPNET_RECORDFILE_INFO 结构数组
	[in]maxlen	nriFileInfo 缓冲的最大长度（单位：字节，建议在（100~200）*sizeof(NET_RECORDFILE_INFO)之间）
	[out]filecount	返回的文件个数，属于输出参数最大只能查到缓冲满为止的录像记录
	[in]waittime	等待时间
	[in]bTime	目前此字段无效
返回值	成功返回 TRUE，失败返回 FALSE	
说明	在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	

表3-36 录像文件类型及卡号对照表

数值	录像文件类型	卡号
0	所有录像文件	NULL
1	外部报警	NULL
2	动态检测报警	NULL
3	所有报警	NULL
4	卡号查询	卡号
5	组合条件查询	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）
6	录像位置与偏移量长度	NULL
8	按卡号查询图片（目前仅 HB-U 和 NVS 特殊型号的设备支持）	卡号
9	查询图片（目前仅 HB-U 和 NVS 特殊型号的设备支持）	NULL
10	按字段查询	FELD1&&FELD2&&FELD3&&（如希望跳过某字段，则相应位置为空）

3.10.2 按时间下载录像 CLIENT_DownloadByTimeEx

表3-37 按时间下载录像 CLIENT_DownloadByTimeEx

选项	说明	
描述	按时间下载录像	
方法	<pre>public LLong CLIENT_DownloadByTimeEx(LLong ILoginID, int nChannelId, int nRecordFileType, NET_TIME tmStart, NET_TIME tmEnd, String sSavedFileName, StdCallCallback cbTimeDownLoadPos, Pointer dwUserData, StdCallCallback fDownLoadDataCallBack, Pointer dwDataUser, Pointer pReserved);</pre>	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 返回值
	[in]nChannelId	设备通道号，从 0 开始
	[in]nRecordFileType	文件查询类型 0: 所有录像文件 1: 外部报警 2: 动态检测录像 3: 所有报警 4: 按卡号查询录像 5: 组合条件查询 8: 按卡号查询图片 9: 查询图片 10: 按字段查询
	[in]tmStart	下载起始时间
	[in]tmEnd	下载结束时间
	[in]sSavedFileName	要保存的录像文件名，全路径
	[in]cbTimeDownLoadPos	下载进度回调函数
	[in]dwUserData	下载进度回调用户自定义数据
	[in]fDownLoadDataCallBack	下载数据回调函数
	[in]dwUserData	下载数据回调用户自定义数据
	[in]pReserved	保留参数，默认为 NULL
返回值	成功返回下载 ID，失败返回 0	
说明	<ul style="list-style-type: none"> NET_IN_PLAY_BACK_BY_TIME_INFO 中回调函数声明 fDataCallBack 和 fDownLoadPosCallBack 请参见“3.11 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 	

3.10.3 停止录像下载 CLIENT_StopDownload

表3-38 停止录像下载 CLIENT_StopDownload

选项	说明	
描述	停止录像下载	
方法	public boolean CLIENT_StopDownload(LLong IFileHandle);	
参数	[in]IFileHandle	CLIENT_DownloadByTimeEx 的返回值
返回值	成功返回下载 ID，失败返回 0	

选项	说明
说明	根据实际需要，等文件下载完成后停止下载，也可以下载到一部分停止下载

3.11 开启实时预览转码接口 CLIENT_RealPlayByDataType

表3-39 CLIENT_RealPlayByDataType

选项	说明	
描述	开启实时预览转码接口	
方法	public LLong CLIENT_RealPlayByDataType(LLong ILoginID,NET_IN_REALPLAY_BY_DATA_TYPE pstInParam,NET_OUT_REALPLAY_BY_DATA_TYPE pstOutParam,int dwWaitTime);	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 返回值
	[in]pstInParam	传入参数结构体
	[out]pstOutParam	传出参数结构体
	[in]dwWaitTime	等待时间
返回值	成功返回下载 ID，失败返回 0	
说明	无	

3.12 开启录像回放转码接口 CLIENT_PlayBackByDataType

表3-40 CLIENT_PlayBackByDataType

选项	说明	
描述	开启录像回放转码接口	
方法	public LLong CLIENT_PlayBackByDataType(LLong ILoginID,NET_IN_PLAYBACK_BY_DATA_TYPE pstInParam,NET_OUT_PLAYBACK_BY_DATA_TYPE pstOutParam,int dwWaitTime);	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 返回值
	[in]pstInParam	传入参数结构体
	[out]pstOutParam	传出参数结构体
	[in]dwWaitTime	等待时间
返回值	成功返回下载 ID，失败返回 0	
说明	无	

3.13 开启录像下载转码接口 CLIENT_DownloadByDataType

表3-41 CLIENT_DownloadByDataType

选项	说明	
描述	开启录像下载转码接口	
方法	public LLong CLIENT_DownloadByDataType(LLong ILoginID,Pointer pstInParam,Pointer pstOutParam,int dwWaitTime);	

选项	说明	
参数	[in]ILoginID	CLIENT_LoginWithHighLevelSecurity 返回值
	[in]pstInParam	传入参数结构体
	[out]pstOutParam	传出参数结构体
	[in]dwWaitTime	等待时间
返回值	成功返回下载 ID，失败返回 0	
说明	无	

3.14 远程通道

3.14.1 获取远程设备信息 CLIENT_MatrixGetCameras

表3-42 获取远程设备信息 CLIENT_MatrixGetCameras

选项	说明	
描述	获取远程设备信息	
方法	public boolean CLIENT_MatrixGetCameras(LLong ILoginID,NET_IN_MATRIX_GET_CAMERAS pInParam,NET_OUT_MATRIX_GET_CAMERAS pOutParam,int nWaitTime);	
参数	[in] ILoginID	CLIENT_LoginWithHighLevelSecurity 返回值
	[in] pInParam	输入参数
	[out] pOutParam	输出参数
	[in] nWaitTime	查询状态等待时间
返回值	<ul style="list-style-type: none"> 成功返回 TRUE 失败返回 FALSE 	
说明	无	

3.14.2 配置远程设备信息 CLIENT_MatrixSetCameras

表3-43 配置远程设备信息 CLIENT_MatrixSetCameras

选项	说明	
描述	配置远程设备信息	
方法	public boolean CLIENT_MatrixSetCameras(LLong ILoginID,NET_IN_MATRIX_SET_CAMERAS pInParam,NET_OUT_MATRIX_SET_CAMERAS pOutParam,int nWaitTime);	
参数	[in] ILoginID	CLIENT_LoginWithHighLevelSecurity 返回值

选项	说明	
	[in] pInParam	输入参数
	[out] pOutParam	输出参数
	[in] nWaitTime	查询状态等待时间
返回值	<ul style="list-style-type: none"> 成功返回 TRUE 失败返回 FALSE 	
说明	无	

第 4 章 回调函数

4.1 断线回调函数 fDisConnect

表4-1 断线回调函数 fDisConnect

选项	说明	
描述	断线回调函数	
接口及方法	<pre>public interface fDisConnect extends StdCallCallback { public void invoke(LLong ILoginID, String pchDVRIP, int nDVRPort, Pointer dwUser); }</pre>	
参数	[out] ILoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[out] pchDVRIP	断线的设备 IP
	[out] nDVRPort	断线的设备端口
	[out] dwUser	回调函数的用户参数
返回值	无	
说明	无	

4.2 断线重连回调函数 fHaveReConnect

表4-2 断线重连回调函数 fHaveReConnect

选项	说明	
描述	断线重连回调函数	
接口及方法	<pre>public interface fHaveReConnect extends StdCallCallback { public void invoke(LLong ILoginID, String pchDVRIP, int nDVRPort, Pointer dwUser); }</pre>	

选项	说明	
参数	[out]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[out]pchDVRIP	断线后重连成功的设备 IP
	[out]nDVRPort	断线后重连成功的设备端口
	[out]dwUser	回调函数的用户参数
返回值	无	
说明	无	

4.3 实时预览数据回调函数 fRealDataCallBackEx

表4-3 实时预览数据回调函数 fRealDataCallBackEx

选项	说明	
描述	实时预览数据回调函数	
接口及方法	<pre>public interface fRealDataCallBackEx extends StdCallCallback { public void invoke(LLong IRealHandle, int dwDataType, Pointer pBuffer, int dwBufSize, int param, Pointer dwUser); }</pre>	
参数	[out]IRealHandle	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	预览数据块地址
	[out]dwBufSize	预览数据块的长度，单位：字节
	[out]param	回调数据参数结构体，dwDataType 值不同类型不同 <ul style="list-style-type: none"> 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	

4.4 音频数据回调函数 pfAudioDataCallBack

表4-4 音频数据回调函数 pfAudioDataCallBack

选项	说明	
描述	语音对讲的音频数据回调函数	
接口及方法	<pre>public interface pfAudioDataCallBack extends StdCallCallback { public void invoke(LLong ITalkHandle, Pointer pDataBuf, int dwBufSize, byte byAudioFlag, Pointer dwUser); }</pre>	
参数	[out]ITalkHandle	CLIENT_StartTalkEx 的返回值

选项	说明	
	[out]pDataBuf	音频数据块地址
	[out]dwBufSize	音频数据块的长度，单位：字节
	[out]byAudioFlag	数据类型标志，0 表示来自本地采集，1 表示来自设备发送
	[out]dwUser	回调函数的用户参数
返回值	无	
说明	无	

4.5 智能事件回调 fAnalyzerDataCallBack

表4-5 智能事件回调 fAnalyzerDataCallBack

选项	说明	
描述	智能事件回调	
接口及方法	<pre>public interface fAnalyzerDataCallBack extends StdCallCallback { public int invoke(LLong IAnalyzerHandle, int dwAlarmType, Pointer pAlarmInfo, Pointer pBuffer, int dwBufSize, Pointer dwUser, int nSequence, Pointer reserved); }</pre>	
参数	[out]IAnalyzerHandle	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
	[out]dwUser	用户数据
	[out]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
	[out]reserved	保留
返回值	无	
说明	无	

4.6 按时间下载回调函数 fTimeDownLoadPosCallBack

表4-6 按时间下载回调函数 fTimeDownLoadPosCallBack

选项	说明
描述	按时间下载回调函数

选项	说明	
接口及方法	<pre>public interface fTimeDownLoadPosCallBack extends StdCallCallback { public void invoke(LLong IPlayHandle, int dwTotalSize, int dwDownLoadSize, int index, NET_RECORDFILE_INFO.ByValue recordfileinfo, Pointer dwUser); }</pre>	
参数	[out]IPlayHandle	CLIENT_DownloadByTimeEx 的返回值
	[out]dwTotalSize	指本次播放总大小，单位：KB
	[out]dwDownLoadSize	指已经播放的大小，单位：KB <ul style="list-style-type: none"> ● -1：表示本次回放结束 ● -2：表示写文件失败
	[out]index	索引
	[out]recordfileinfo	录像文件信息
	[out]dwUser	用户数据
返回值	无	
说明	无	

4.7 报警订阅回调函数 fMessCallBack

表4-7 报警订阅回调函数 fMessCallBack

选项	说明	
描述	实时预览数据回调函数	
接口及方法	<pre>public interface fMessCallBack extends StdCallCallback{ public boolean invoke(int ICommand , LLong ILoginID , Pointer pStuEvent , int dwBufLen , String strDeviceIP , NativeLong nDevicePort , Pointer dwUser); }</pre>	
参数	[out]ICommand	具体事件类
	[out]ILoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[out]pStuEvent	返回的数据指针
	[out]dwBufLen	返回的指针的长度
	[out]strDeviceIP	返回的 ip
	[out]nDevicePort	返回的端口号
	[out]dwUser	回调函数的用户参数
返回值	无	
说明	无	

4.8 异步抓图回调 fSnapRev

表4-8 异步抓图回调

选项	说明	
描述	异步抓图回调函数	
接口及方法	<pre>public interface fSnapRev extends Callback{ public void invoke(LLong ILoginID , Pointer pBuf, int RevLen, int EncodeType, int CmdSerial, Pointer dwUser); }</pre>	
参数	[out]ILoginID	CLIENT_LoginWithHighLevelSerity 的返回值
	[out]pBuf	异步抓图图片地址
	[out]RevLen	异步抓图图片的长度
	[out]EncodeType	编码类型
	[out]CmdSerial	操作流水号
	[out]dwUser	回调函数的用户参数
返回值	无	
说明	无	

4.9 实时预览转码数据回调函数 fDataCallBackEx

表4-9 实时预览转码数据回调

选项	说明	
描述	实时预览转码数据回调函数原型扩展 2	
接口及方法	<pre>public interface fRealDataCallBackEx extends SDKCallback{ public void invoke(LLong IRealHandle, int dwDataType, Pointer pBuffer, int dwBufSize, int param, Pointer dwUser); }</pre>	
参数	[out]IRealHandle,	CLIENT_RealPlayByDataType 的返回值
	[out]dwDataType	0-原始数据 1-帧数据 2-yuv 数据 3-pcm 音频数据
	[out]pBuffer	对应 BYTE 数据
	[out]dwBufSize	BYTE 长度
	[out]param	当类型为 0(原始数据)和 2(YUV 数据) 时为 0。当回调的数据类型为 1 时 param 为一个 tagVideoFrameParam 结构体指针。当数据类型是 3 时 ,param 也是一个 tagCBPCMDDataParam 结构体指针
	[out]dwUser	回调函数的用户参数
返回值	无	
说明	无	

4.10 回放进度回调函数 fDownloadPosCallBack

表4-10 fDownloadPosCallBack

选项	说明	
描述	回放进度回调函数	
接口及方法	<pre>public interface fDownloadPosCallBack extends SDKCallback { public void invoke(LLong IPlayHandle, int dwTotalSize, int dwDownloadSize, Pointer dwUser); }</pre>	
参数	[out]IPlayHandle	CLIENT_PlayBackByDataType 的返回值
	[out]dwTotalSize	指本次播放总大小，单位：KB
	[out]dwDownloadSize	指已经播放的大小，单位：KB -1：表示本次回放结束 -2：表示写文件失败
	[out]dwUser	回调函数的用户参数
返回值	无	
说明	录像回放时，回放进度回调函数。 在该回调函数中不建议调用任何 NetSDK 接口。但如果 Demo 内该回调函数调用了 NetSDK 接口，则您可以同样处理。	


4.11 回放数据回调 fDataCallBack

表4-11 fDataCallBack

选项	说明	
描述	回放进度回调函数	
接口及方法	<pre>public interface fDataCallBack extends SDKCallback { public void invoke(LLong IPlayHandle, int dwDataType, Pointer pBuffer, int dwBufSize, Pointer dwUser); }</pre>	
参数	[out]IPlayHandle	CLIENT_PlayBackByDataType 的返回值
	[out]dwDataType	配合 EM_REAL_DATA_TYPE 使用,码流转换后的数据回调函数 (fRealDataCallBackEx,fDataCallBack)中的参数 dwDataType 的值
	[out]pBuffer	对应 BYTE 数据
	[out]dwBufSize	BYTE 长度
	[out]dwUser	回调函数的用户参数
返回值	无	
说明	录像回放时，回放进度回调函数。 在该回调函数中不建议调用任何 NetSDK 接口。但如果 Demo 内该回调函数调用了 NetSDK 接口，则您可以同样处理。	

附录1 法律声明

商标声明

-  : 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。