

# NetSDK\_Python 编程指导手册



# 前言

## 目的

欢迎使用 NetSDK Python 版本编程指导手册。











本文档主要描述 Python 的 Demo 中的一些功能、接口以及函数之间的调用关系，并提供了代码示例。

## 读者对象

使用 NetSDK 的软件开发工程师、产品经理、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
 <b>危险</b>	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。
 <b>警告</b>	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。
 <b>注意</b>	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。
 <b>防静电</b>	表示静电敏感的设备。
 <b>当心触电</b>	表示高压危险。
 <b>激光辐射</b>	表示强激光辐射。
 <b>风扇警告</b>	表示危险运动部件，请远离运动风扇叶片。
 <b>当心机械伤人</b>	表示设备部件机械伤人。
 <b>窍门</b>	表示能帮助您解决某个问题或节省您的时间。
 <b>说明</b>	表示是正文的附加信息，是对正文的强调和补充。

## 修订记录

版本号	修订内容	发布日期
V1.0.1	将封装库的打包方式更改为 whl 安装包形式。	2020.10
V1.0.0	首次发布。	2020.05

# 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。

接口	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。
分辨率	分辨率包括显示分辨率和图像分辨率。显示分辨率指单位面积显示像素的数量；图像分辨率指图像中存储的信息量，指每英寸图像内有多少个像素点。
视频通道	<b>NetSDK</b> 与设备通信、视频流传输的抽象概念。如存储设备（如 <b>NVR</b> ）挂载若干前端设备（ <b>SD</b> ， <b>IPC</b> 等），存储设备（ <b>NVR</b> ）将前端设备（ <b>SD</b> ， <b>IPC</b> 等）抽象为视频通道进行管理，通道号从 0 开始。若 <b>NetSDK</b> 与单个前端设备直连，则视频通道号一般为 0。
动态检测报警	检测到画面上有移动物体时，会有动态检测报警上报。

# 目录

前言.....	I
名词解释.....	II
第 1 章 内容简介 .....	1
1.1 概述 .....	1
1.2 环境要求.....	1
1.3 运行 Demo.....	2
1.3.1 安装 whl 文件 .....	2
1.3.2 运行 Demo .....	3
1.4 工程配置 .....	4
1.4.1 pycharm 配置.....	4
1.4.2 添加工具到 pycharm .....	6
第 2 章 主要功能 .....	12
2.1 NetSDK 初始化 .....	12
2.1.1 简介 .....	12
2.1.2 接口总览.....	12
2.1.3 流程说明.....	12
2.1.4 示例代码.....	13
2.1.5 注意事项.....	13
2.2 设备搜索及初始化设备账户 .....	14
2.2.1 简介 .....	14
2.2.2 接口总览.....	14
2.2.3 流程说明.....	15
2.2.4 示例代码.....	17
2.3 设备登录 .....	20
2.3.1 简介 .....	20
2.3.2 接口总览.....	20
2.3.3 流程说明.....	21
2.3.4 示例代码.....	21
2.3.5 注意事项.....	22
2.4 实时监控 .....	22
2.4.1 简介 .....	22
2.4.2 接口总览.....	23
2.4.3 流程说明.....	23
2.4.4 示例代码.....	24
2.4.5 注意事项.....	24
2.5 录像回放 .....	25
2.5.1 简介 .....	25
2.5.2 接口总览.....	25
2.5.3 流程说明.....	25
2.5.4 示例代码.....	26
2.6 录像下载 .....	28
2.6.1 简介 .....	28
2.6.2 接口总览.....	28

2.6.3 流程说明.....	28
2.6.4 示例代码.....	29
2.7 设备控制.....	31
2.7.1 简介.....	31
2.7.2 接口总览.....	31
2.7.3 流程说明.....	31
2.7.4 示例代码.....	32
2.8 远程抓图.....	33
2.8.1 简介.....	33
2.8.2 接口总览.....	33
2.8.3 流程说明.....	34
2.8.4 示例代码.....	35
2.9 报警上报.....	35
2.9.1 简介.....	35
2.9.2 接口总览.....	35
2.9.3 流程说明.....	36
2.9.4 示例代码.....	37
2.10 智能事件上报.....	38
2.10.1 简介.....	38
2.10.2 接口总览.....	38
2.10.3 流程说明.....	38
2.10.4 示例代码.....	39
<b>第 3 章 接口函数.....</b>	<b>42</b>
3.1 NetSDK 初始化.....	42
3.1.1 NetSDK 初始化 InitEx.....	42
3.1.2 NetSDK 清理 Cleanup.....	42
3.1.3 设置断线重连回调函数 SetAutoReconnect.....	42
3.2 设备搜索及初始化设备账户.....	43
3.2.1 异步搜索设备 StartSearchDevicesEx.....	43
3.2.2 跨网段搜索设备 SearchDevicesByIPs.....	43
3.2.3 停止异步搜索同网段内设备 StopSearchDevices.....	44
3.2.4 初始化设备账户 InitDevAccount.....	44
3.3 设备登录.....	45
3.3.1 用户登录设备 LoginWithHighLevelSecurity.....	45
3.3.2 用户登出设备 Logout.....	45
3.4 实时监视.....	46
3.4.1 打开监视 RealPlayEx.....	46
3.4.2 关闭监视 StopRealPlay.....	47
3.5 录像回放.....	47
3.5.1 设置工作模式 SetDeviceMode.....	47
3.5.2 查询时间段内的所有录像文件 QueryRecordFile.....	48
3.5.3 按时间方式回放 PlayBackByTimeEx2.....	48
3.5.4 停止录像回放 StopPlayBack.....	49
3.5.5 暂停或继续录像回放 PausePlayBack.....	49
3.6 录像下载.....	49
3.6.1 按时间下载录像 DownloadByTimeEx.....	49
3.6.2 停止录像下载 StopDownload.....	50

3.7 设备控制 .....	51
3.7.1 获取配置信息函数 GetDevConfig .....	51
3.7.2 设置配置信息函数 SetDevConfig .....	51
3.7.3 远程设备重启函数 RebootDev .....	52
3.8 远程抓图 .....	52
3.8.1 设置远程抓图数据回调函数 SetSnapRevCallBack .....	52
3.8.2 抓图请求扩展接口 SnapPictureEx .....	52
3.9 报警上报 .....	53
3.9.1 设置报警回调函数接口 SetDVRMessCallBackEx1 .....	53
3.9.2 订阅报警扩展接口 StartListenEx .....	53
3.9.3 停止订阅报警接口 StopListen .....	54
3.10 智能事件上报 .....	54
3.10.1 智能图片报警订阅接口 RealLoadPictureEx .....	54
3.10.2 停止智能事件订阅接口 StopLoadPic .....	55
<b>第 4 章 回调函数定义 .....</b>	<b>56</b>
4.1 断线回调函数 fDisConnect .....	56
4.2 断线重连回调函数 fHaveReConnect .....	56
4.3 异步搜索设备回调函数 fSearchDevicesCBEx .....	57
4.4 搜索设备回调函数 fSearchDevicesCB .....	57
4.5 回放进度回调函数 fDownLoadPosCallBack .....	57
4.6 回放数据回调 fDataCallBack .....	58
4.7 按时间下载进度回调函数 fTimeDownLoadPosCallBack .....	58
4.8 智能图片报警回调函数 fAnalyzerDataCallBack .....	59
4.9 抓图回调函数 fSnapRev .....	60
4.10 报警上报回调函数 fMessCallBackEx1 .....	60
<b>附录 1 法律声明 .....</b>	<b>62</b>
<b>附录 2 网络安全建议 .....</b>	<b>63</b>

# 第 1 章 内容简介

## 1.1 概述

本开发套件主要包括以下功能：

设备登录、实时监控、录像回放、录像下载、远程抓图、报警上报、设备搜索、智能事件上报与抓图、设备重启、设备校时等。

- Python 的 NetSDK 库所包含的文件，请参见表 1-1。

表1-1 NetSDK 库包括的文件

库类型	库文件名称	库文件说明
功能库	dhnetsdk.dll	库文件
	avnetsdk.dll	库文件
配置库	dhconfigsdk.dll	库文件
播放（编码解码）辅助库	dhplay.dll	播放库
	fisheye.dll	鱼眼矫正库
avnetsdk.dll 的依赖库	Infra.dll	基础库
	json.dll	json 库
	NetFramework.dll	网络基础库
	Stream.dll	媒体传输结构体封装库
	StreamSvr.dll	流服务
dhnetsdk 辅助库	lvsDrawer.dll	图像显示库

- Python 的封装工程所包含的文件，请参见表 1-2。

表1-2 NetSDK 文件夹包括的文件

文件名称	文件说明
NetSDK.py	调用 NetSDK 库，将库内的接口封装为 Python 接口供用户使用
SDK_Callback.py	存放所用 NetSDK 库用到的回调函数
SDK_Enum.py	存放所用 NetSDK 库用到的枚举
SDK_Struct.py	存放所用 NetSDK 库用到的结构体

### 说明

- NetSDK 的功能库和配置库是必备库。
- 功能库是设备网络 NetSDK 的主体，主要用于网络客户端与各类产品之间的通讯交互，负责远程控制、查询、配置及码流数据的获取和处理等。
- NetSDK 库是 Python 封装工程的基础，工程中 NetSDK.py 文件内定义了 NetSDK 库的引用路径，最终使用时请将 NetSDK 库放到相应路径下。用户可自定义引用路径。
- 所有对外使用的接口，全部定义在 NetClient 类中，使用时需要先定义一个 NetClient 类的对象，通过对象来调用类中的各个接口。

## 1.2 环境要求

- 推荐内存：不低于 512M。

- 推荐 Python 使用版本：3.7 及以上版本。
- 操作系统：
  - ◇ Windows: Windows 10/Windows 8.1/Windows 7/2000 以及 Windows Server 2008/2003。
  - ◇ Linux: Red Hat/SUSE 等通用 Linux 系统。

## 1.3 运行 Demo

下载并解压 Python 版 NetSDK 开发包，找到 dist 文件夹下的.whl 文件。根据系统不同，对应名称可能略有区别，如 “NetSDK-2.0.0.1-py3-none-win\_amd64.whl” 或 “NetSDK-2.0.0.1-py3-none-linux\_i686.whl”。

此文件是 NetSDK 封装库的 python 安装包。安装.whl 文件后，在 Demo 开始处插入 “import NetSDK”，即可在后续开发中使用封装库内容。

### 1.3.1 安装 whl 文件

#### 操作步骤

- 步骤1 安装 python3.7，并将安装目录添加到系统环境变量中。
- 步骤2 启动命令终端，运行以下命令来安装 pyqt5 和 pyqt5-tools。

```
pip install pyqt5
pip install pyqt5-tools
```

- 步骤3 在 NetSDK 封装库.whl 文件存放目录下打开命令终端，运行以下命令安装插件。

```
pip install NetSDK-2.0.0.1-py3-none-win_amd64.whl
```

#### 注意事项

- Windows 操作系统下，安装文件位于 Python 安装目录的 “\Lib\site-packages” 路径下的 “NetSDK” 文件夹内。Linux 操作系统下，安装文件位于 Python 安装目录的 “site-packages” 下的 NetSDK 文件夹内。如果需要参考或者修改其中的内容，请参考该目录下文件。 “site-packages” 目录存放了用户自定义安装的插件和安装的 PyQt 等。
- 如果需要卸载插件，请使用 “pip uninstall NetSDK” 命令。
- 如果系统中同时存在 python2 和 python3，请将命令中的 “pip” 替换为 “pip3”。
- whl 安装后，即可使用 “import NetSDK” 开发 SDK 相关功能，客户自行开发的程序可不用依赖于 PyQt。
- 如果处于离线环境下，运行以上命令无法安装成功。请前往 python 官网的 pypi 模块 (<https://pypi.org/>) 下载以下插件，并根据系统和 python 的版本，安装对应版本的插件。插件按顺序包括：python\_dotenv、click、PyQt5-sip、PyQt5、pyqt5-tools、PyQt5Designer。
- 离线安装插件时，在插件所在目录打开命令终端，运行 “pip install xxx” 的命令。本地演示时使用的命令如下，请参考(Linux 除插件名称可能不同外，和 Windows 下没区别)：

```
pip install python_dotenv-0.10.1-py2.py3-none-any.whl
pip install Click-7.0-py2.py3-none-any.whl
pip install PyQt5_sip-4.19.13-cp37-none-win_amd64.whl
pip install PyQt5-5.11.3-5.11.2-cp35.cp36.cp37.cp38-none-win_amd64.whl
```



```
pip install pyqt5_tools-5.11.3.1.4-cp37-none-win_amd64.whl
pip install PyQt5Designer-5.10.1-cp37-none-win_amd64.whl
```

### 1.3.2 运行 Demo

Whl 打包程序安装完成后，直接运行 Demo。

以运行实时监视 Demo 为例。进入 “RealPlayDemo” 文件夹，启动命令终端，运行 “python RealPlayDemo.py” 命令，demo 就会启动。

Windows 下，如果 py 文件是以 python 的方式打开，直接双击 RealPlayDemo.py 文件也可启动程序。

### 注意事项

- 如果系统中同时存在 python2 和 python3，请将命令 “python RealPlayDemo.py” 替换为 “python3 RealPlayDemo.py”。
- Windows 下，双击文件运行 demo，会在后方出现一个多余的控制台窗口。如果希望在运行程序时隐藏控制台框口，可将 RealPlayDemo.py 的后缀改为 “.pyw”，即改成 “RealPlayDemo.pyw”，再双击运行。
- 使用 PyCharm 开发时，无需打开整个目录，打开 Demo 文件夹下的各个 Demo 目录即可。

## 1.4 工程配置

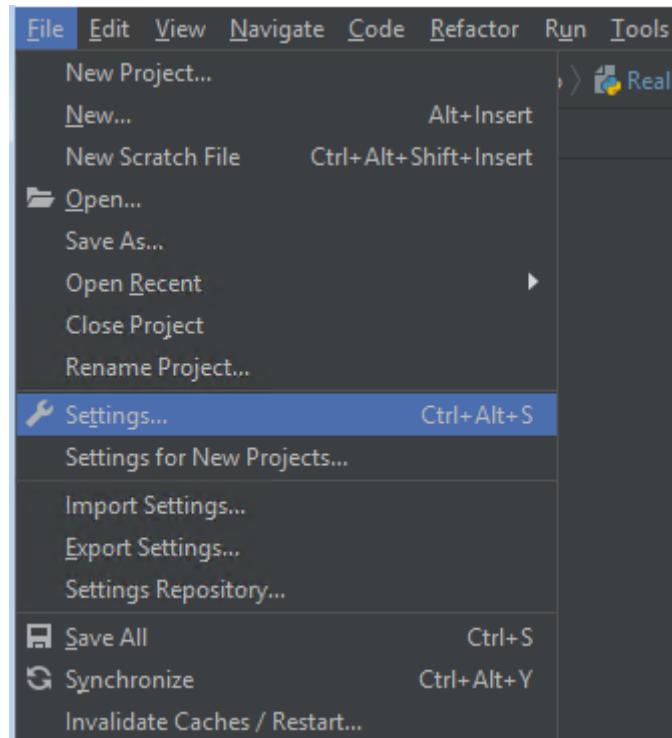
### 1.4.1 pycharm 配置

需要先配置 Interpreter，才能使用 pycharm 运行 Demo 工程。

步骤1 打开 pycharm 软件。

步骤2 选择 “File > Settings”。

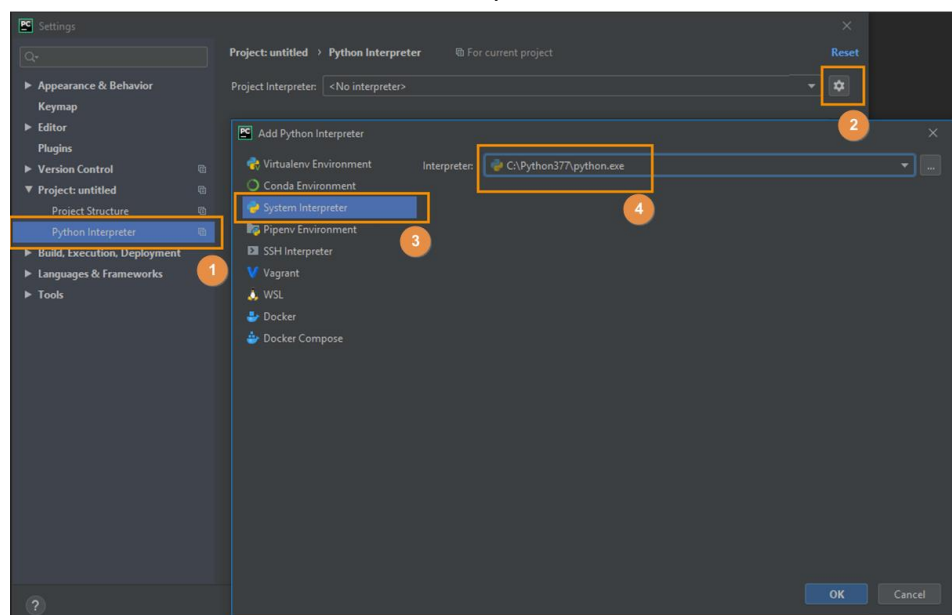
图1-1 选择 Settings



步骤3 配置 Interpreter。

软件显示 PyQt5 相关软件的信息。

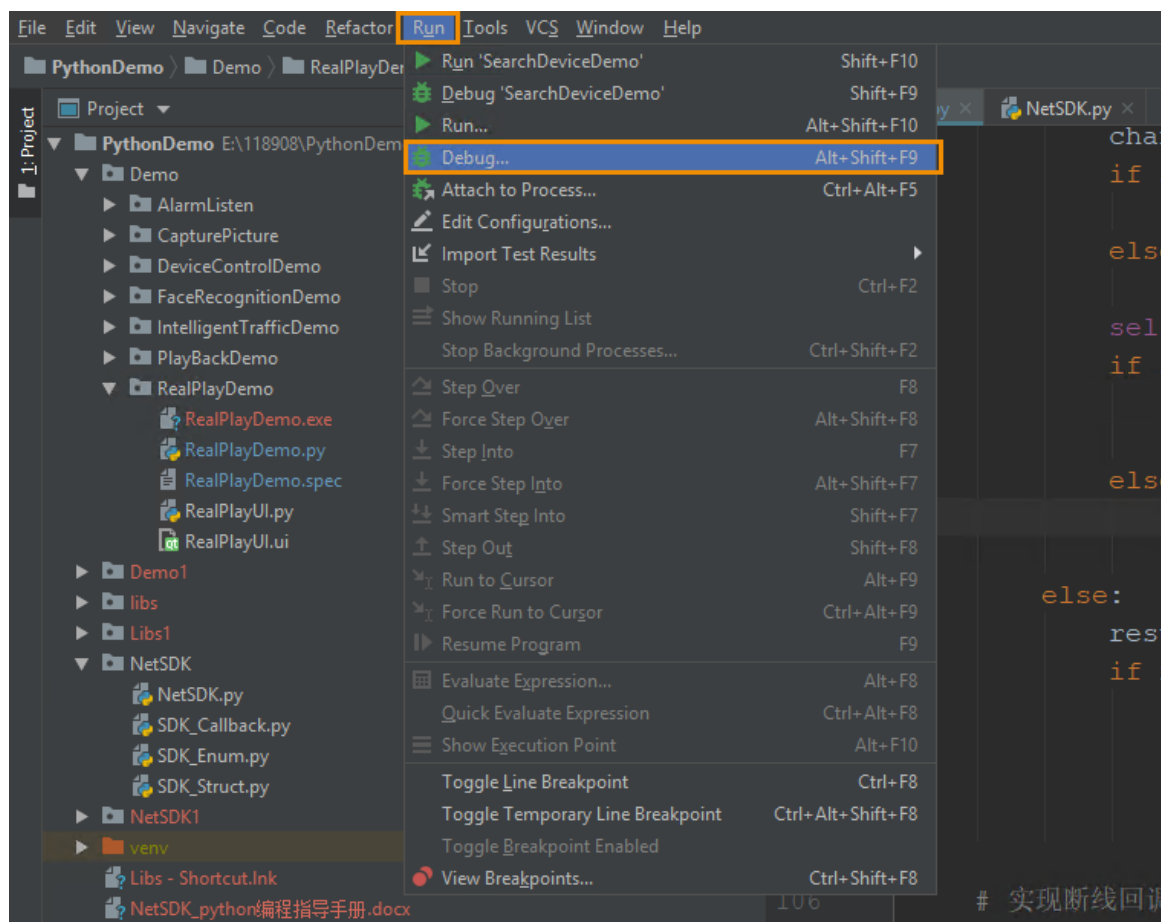
图1-2 配置 Interpreter



#### 步骤4 配置 Demo。

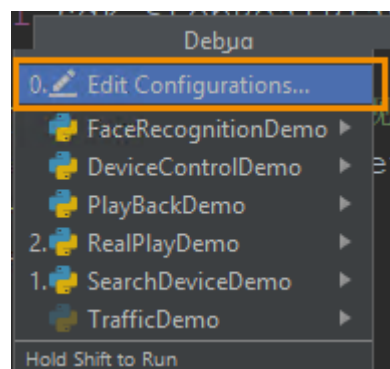
1. 选择 “Run > Debug”。

图1-3 配置 Demo (1)



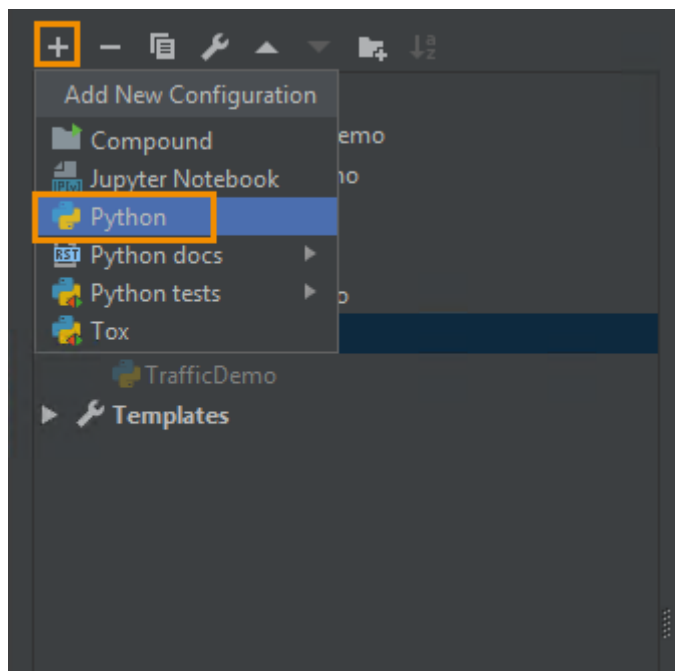
2. 在弹出对话框单击 “Edit Configurations”。

图1-4 配置 Demo (2)



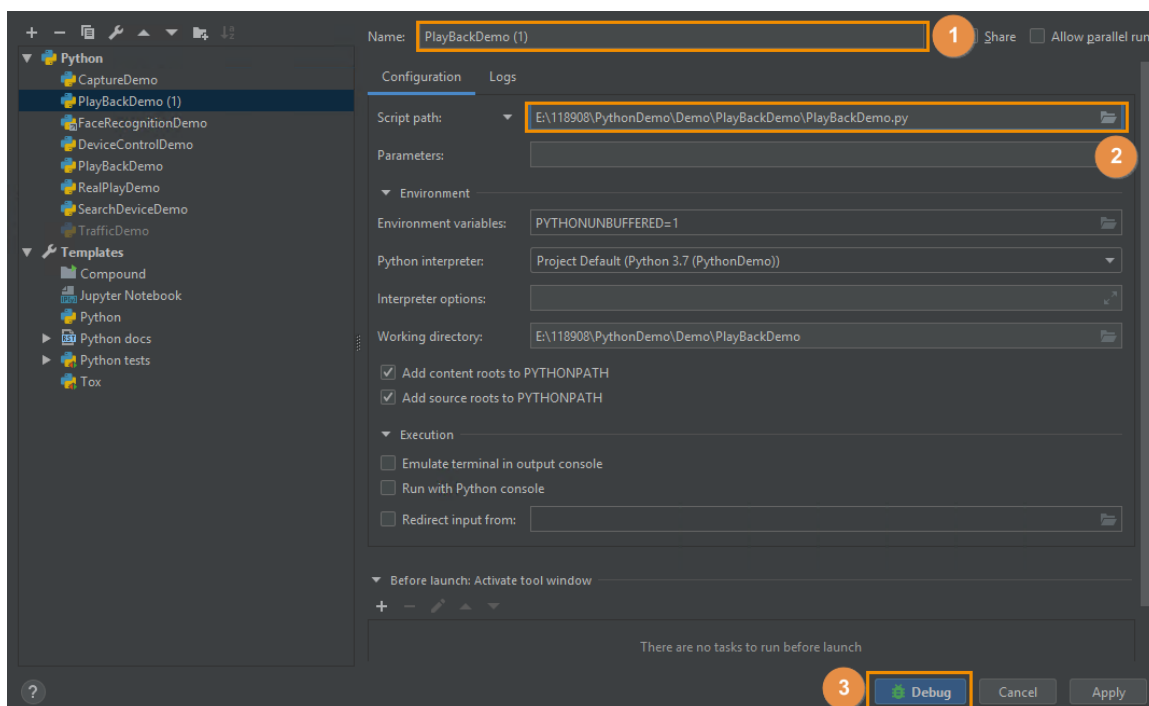
3. 选择 “+ > Python”。

图1-5 配置 Demo (3)



4. 设置 Demo 配置名称和 Demo.py 所在路径。
  - ◇ Name: 设置 Demo 配置名称。
  - ◇ Script path: 选择 Demo.py 所在路径。本文以 PlayBackDemo.py 举例。
5. 单击“Debug”，运行 Demo。

图1-6 配置 Demo 配置



## 1.4.2 添加工具到 pycharm

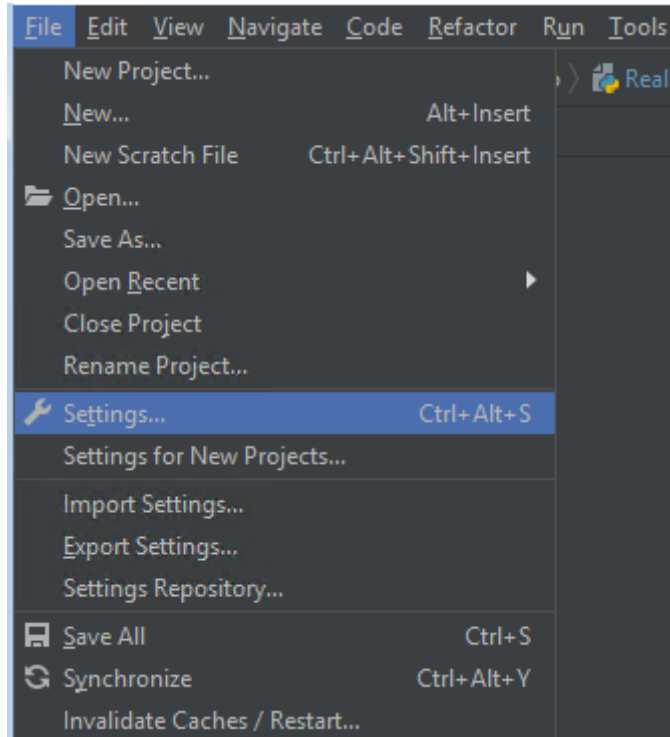
添加 pyqt5designer 和 pyuic5 工具到 pycharm。

- 添加 pyqt5designer 工具到 pycharm 后，选中对应 ui 文件，即可跳转到 qt designer 软件，使用该软件设计 UI 界面。

- 添加 pyuic5 工具到 pycharm 后, 选中对应 ui 文件, 生成 py 文件, 通过 py 文件即可查看定义的变量。

步骤1 选择 “File > Settings” 。

图1-7 进入设置界面



步骤2 添加 pyqt5designer 工具。选择 “Tool > External Tools”，单击 “+”，在弹出的对话框配置 pyqt5designer 工具参数，单击 “OK” 。

图1-8 添加 pyqt5designer 工具

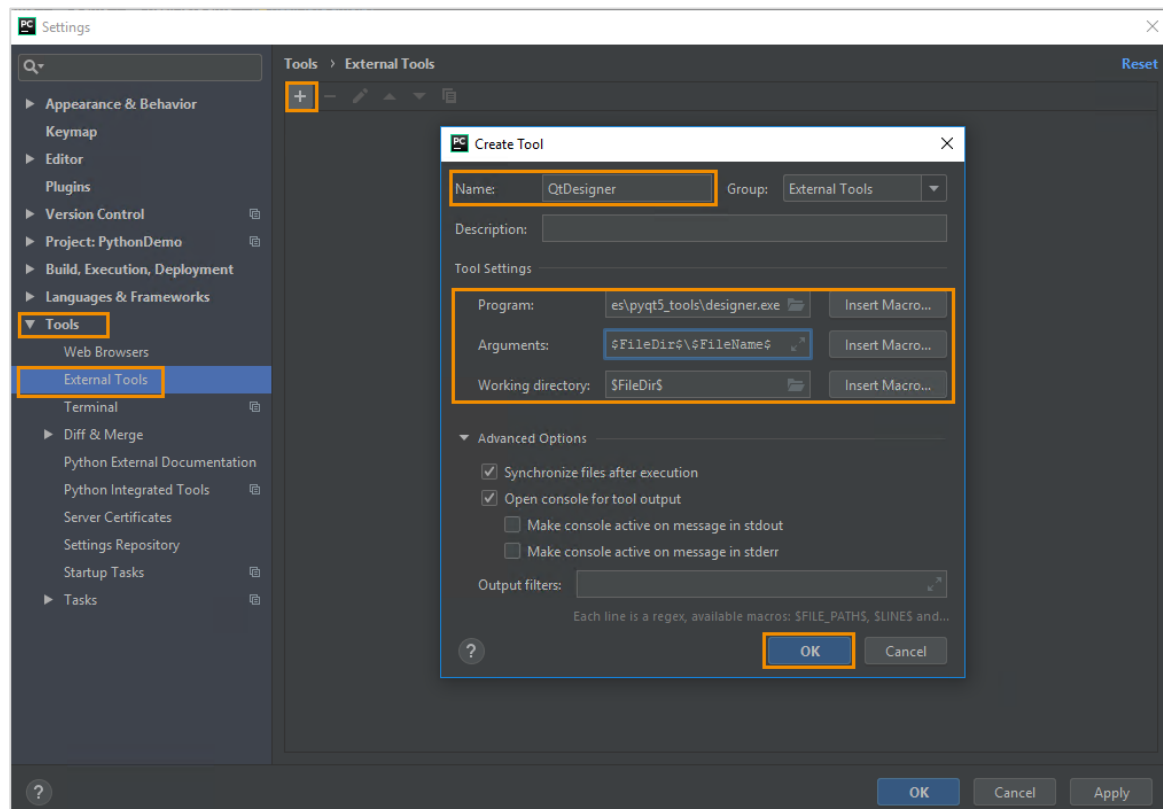


表1-3 添加 pyqt5designer 工具参数说明

参数	说明
Name	工具名称，用户可以自定义，例如 QtDesigner。
Program	填写 Python 安装目录 Scripts 文件夹下的 pyqt5designer.exe 的路径。
Arguments	\$FileDir\$\FileName\$
Working directory	\$FileDir\$

步骤3 添加 pyuic5 工具。单击“+”，在弹出的对话框配置 pyuic5 工具参数，单击“OK”。

图1-9 添加 pyuic5 工具

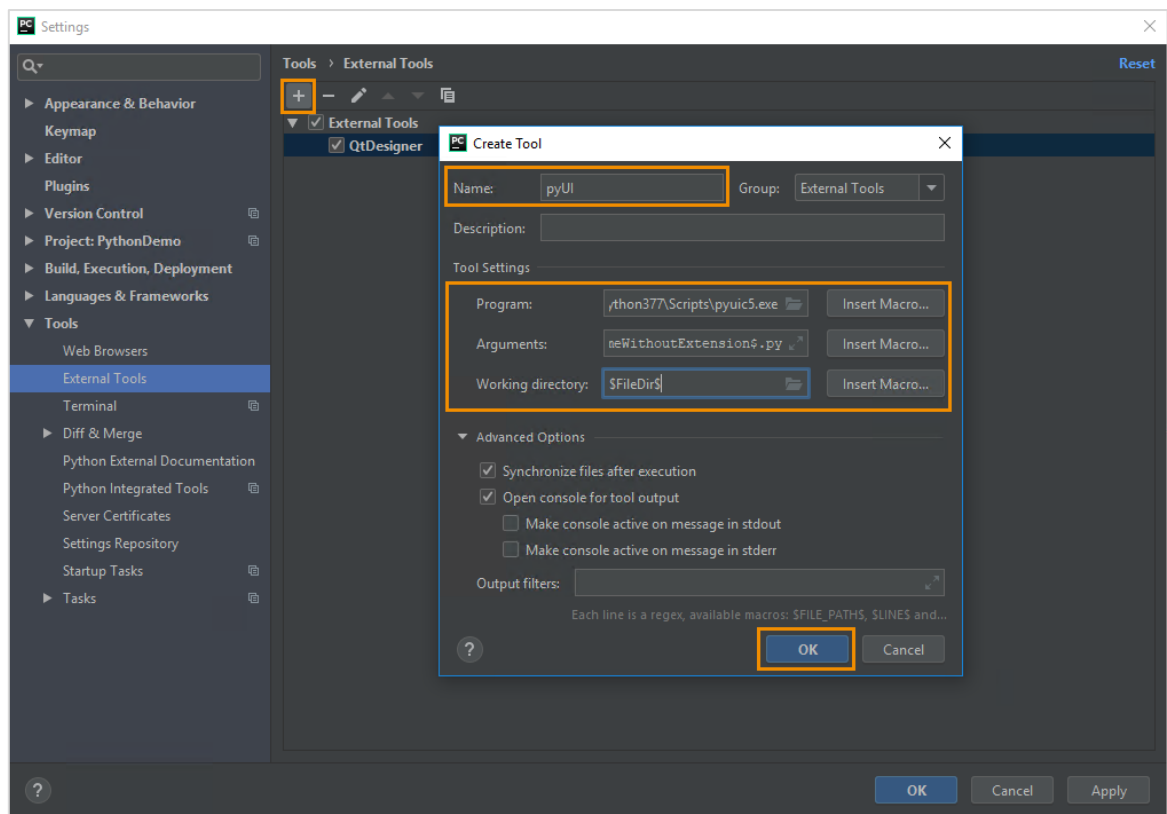


表1-4 添加 pyuic5 工具参数说明

参数	说明
Name	工具名称，用户可以自定义，例如 PyUI。
Program	填写 Python 安装目录 Scripts 文件夹下的 pyuic5.exe 的路径。
Arguments	\$FileName\$ -o \$FileNameWithoutExtension\$.py
Working directory	\$FileDir\$

步骤4 使用 QtDesigner 工具设计界面。选中对应的.ui 文件，在右键菜单栏选择“External Tools > QtDesigner”，打开 QtDesigner，即可设计界面。

图1-10 打开 QtDesigner 工具

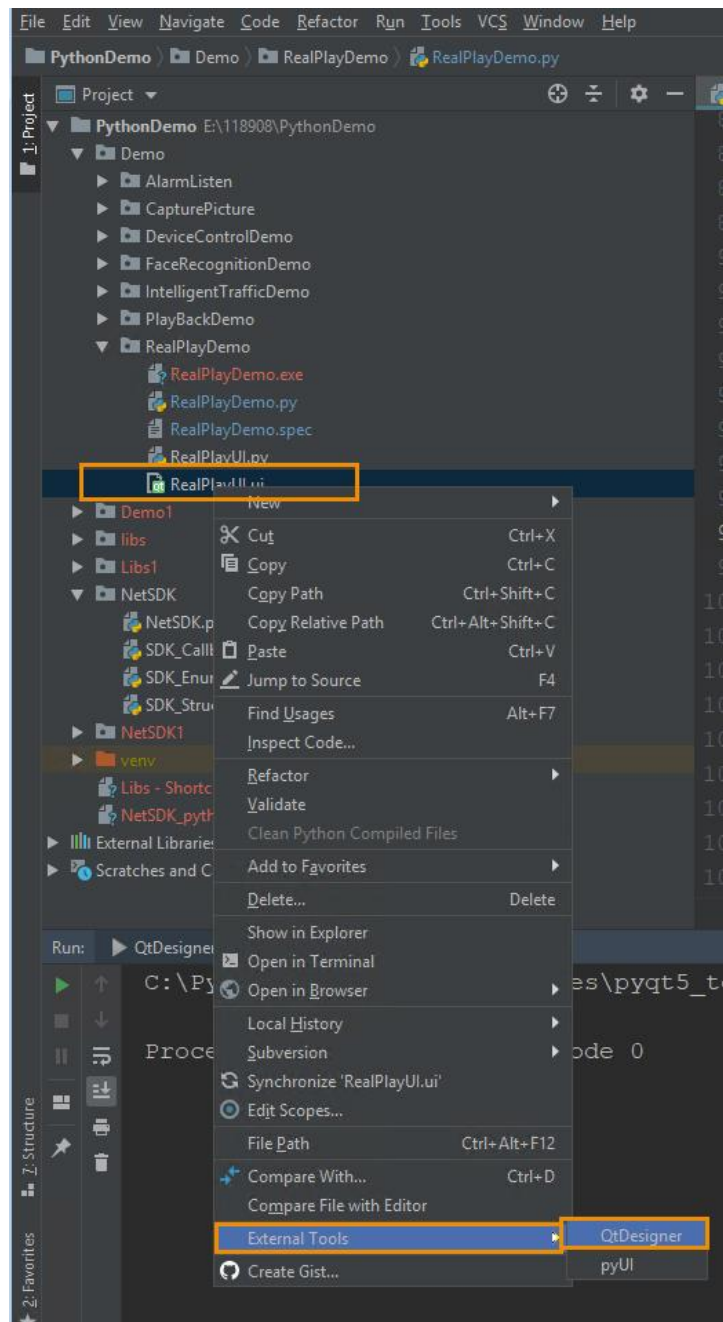
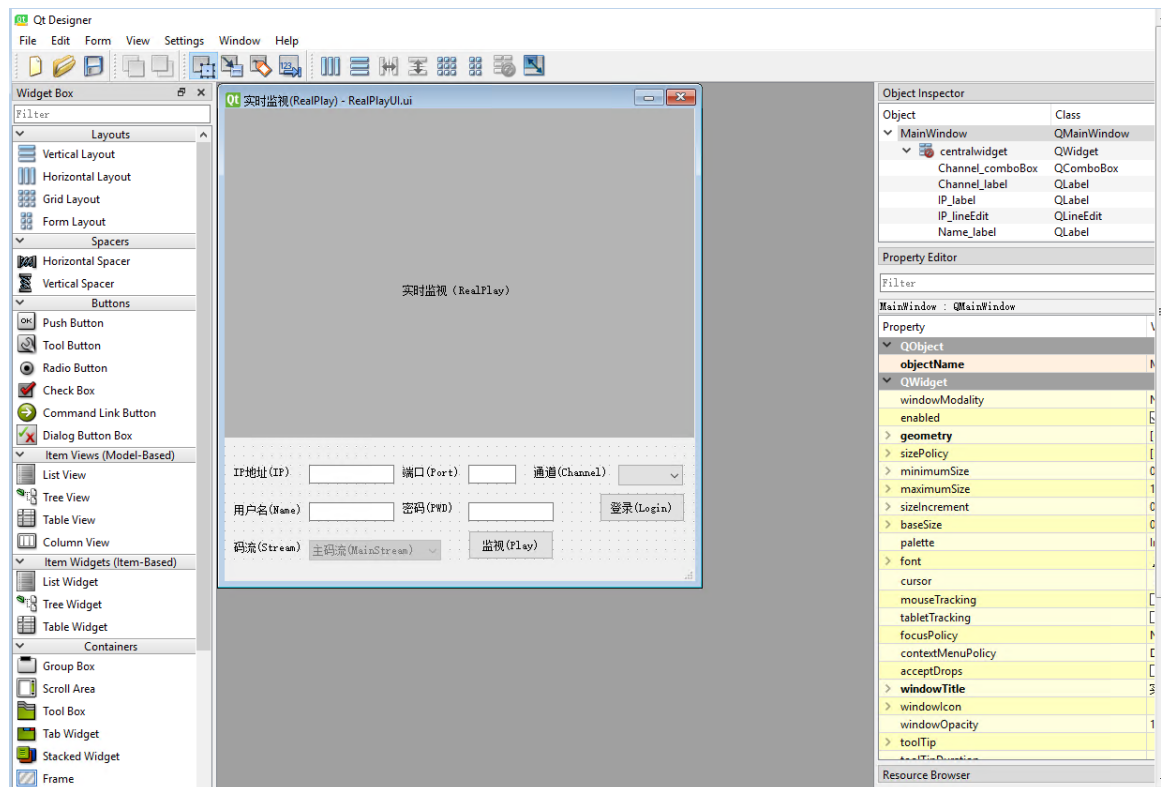


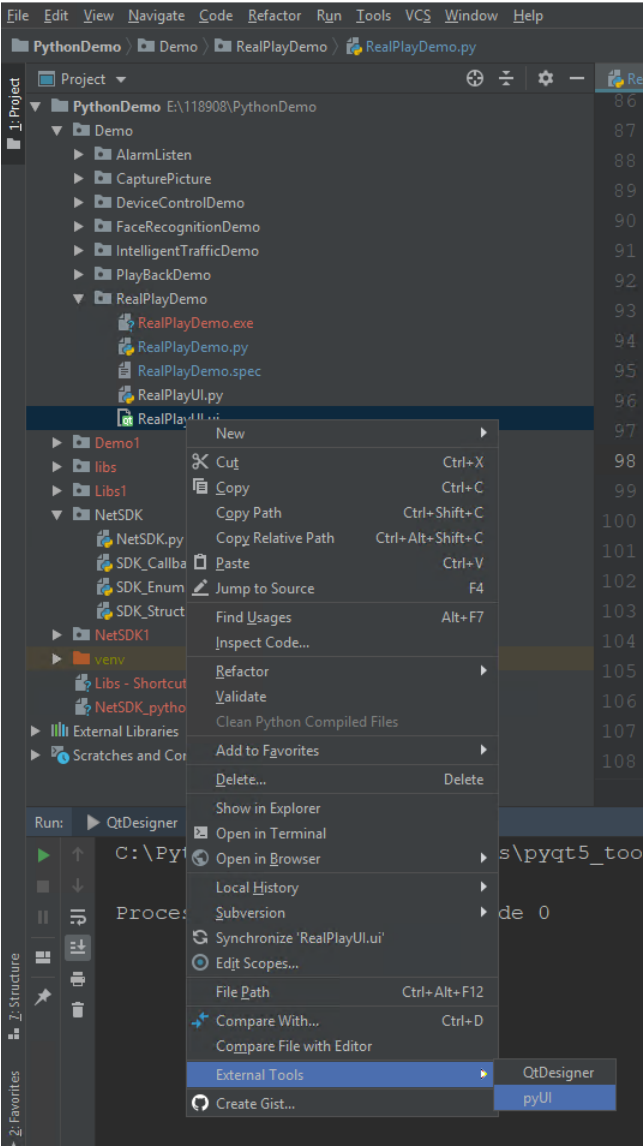
图1-11 使用 QtDesigner 设计界面



步骤5 将.ui 文件转换成.py 文件。选中对应的.ui 文件，在右键菜单栏选择“External Tools > pyuic5”，将当前的.ui 文件转换成.py 文件。



图1-12 将.ui 文件转换成.py 文件



# 第 2 章 主要功能

## 2.1 NetSDK 初始化

### 2.1.1 简介

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

- NetSDK 的初始化将会占用一定的内存。
- 同一个进程内，只有第一次初始化有效。
- 使用完毕后需要调用 NetSDK 清理接口以释放资源。
- InitEx 和 Cleanup 接口一一对应，在编写代码时，建议仅调用一次。

### 2.1.2 接口总览

表2-1 NetSDK 初始化接口说明

接口	说明
_load_library	加载动态库
InitEx	NetSDK 初始化接口
SetAutoReconnect	设置断线重连回调（可选）
Cleanup	释放 NetSDK 资源。

### 2.1.3 流程说明

图2-1 NetSDK 初始化业务流程



## 注意事项

**InitEx** 只需要在整个 **Demo** 的开始、使用 **NetSDK** 接口前调用一次，**Cleanup** 只需要在所有与 **NetSDK** 相关功能都使用完毕后调用一次释放 **NetSDK** 资源即可。这两个接口不需要每用一个功能都调用一次。

## 流程说明

- 步骤1 调用 **\_load\_library**，加载动态库。
- 步骤2 调用 **InitEx**，初始化 **NetSDK** 并设置断线回调函数。
- 步骤3 （可选）调用 **SetAutoReconnect**，设置断线重连成功回调函数。
- 步骤4 调用 **Cleanup**，释放 **NetSDK** 资源。**NetSDK** 功能使用完后才能调用该接口。

### 2.1.4 示例代码

```
# 声明并初始化回调函数
self.m_DisConnectCallBack = fDisConnect(self.DisConnectCallBack)
self.m_ReConnectCallBack = fHaveReConnect(self.ReConnectCallBack)

# 获取 NetSDK 对象并初始化
self.sdk = NetClient()
self.sdk.InitEx(self.m_DisConnectCallBack)
self.sdk.SetAutoReconnect(self.m_ReConnectCallBack)

# 实现断线回调函数功能
def DisConnectCallBack(self, lLoginID, pchDVRIP, nDVRPort, dwUser):
    self.setWindowTitle("实时监控(RealPlay)-断线(OffLine)")

# 实现断线重连回调函数功能
def ReConnectCallBack(self, lLoginID, pchDVRIP, nDVRPort, dwUser):
    self.setWindowTitle("实时监控(RealPlay)-重新连接(OnLine)")

# 释放 NetSDK 资源
self.sdk.Cleanup()
```

### 2.1.5 注意事项

- **InitEx** 只需要在整个 **Demo** 的开始、使用 **NetSDK** 接口前调用一次，**Cleanup** 只需要在所有与 **NetSDK** 相关功能都使用完毕后调用一次释放 **NetSDK** 资源即可。这两个接口不需要每用一个功能都调用一次。
- **\_load\_library** 是 **NetClient** 类的内部函数，在实现 **NetClient** 类对象时会自动调用，用户无需手动调用。此处显式说明，只是为了提示用户，如果用户想要改变 **NetSDK** 库位置或者改变调用 **NetSDK** 库的方式和时机，可以修改这个函数。
- 初始化：**InitEx** 接口在应用程序开始使用 **sdk** 接口前调用，无需多次调用。
- 清理：**Cleanup** 接口会清理所有已开启的业务，如登录、实时监控和报警订阅等。

- 断线重连：NetSDK 可以设置断线重连功能，当遇到一些特殊情况（例如断网、断电等）设备断线时，在 NetSDK 内部会定时持续不断地进行登录操作，直至成功登录设备。断线重连后可以恢复实时监视、录像回放业务、智能事件订阅和报警订阅，其他业务无法恢复。
- 关于示例代码中回调函数的详细介绍，请参见“第 4 章回调函数定义”。

## 2.2 设备搜索及初始化设备账户

### 2.2.1 简介

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

设备搜索根据是否跨网段可分为以下两种：

- 异步同网段设备搜索。搜索当前操作所在网段内的设备信息。
- 同步跨网段设备搜索。根据用户设置的网段信息，搜索相应网段内的设备信息。

### 2.2.2 接口总览

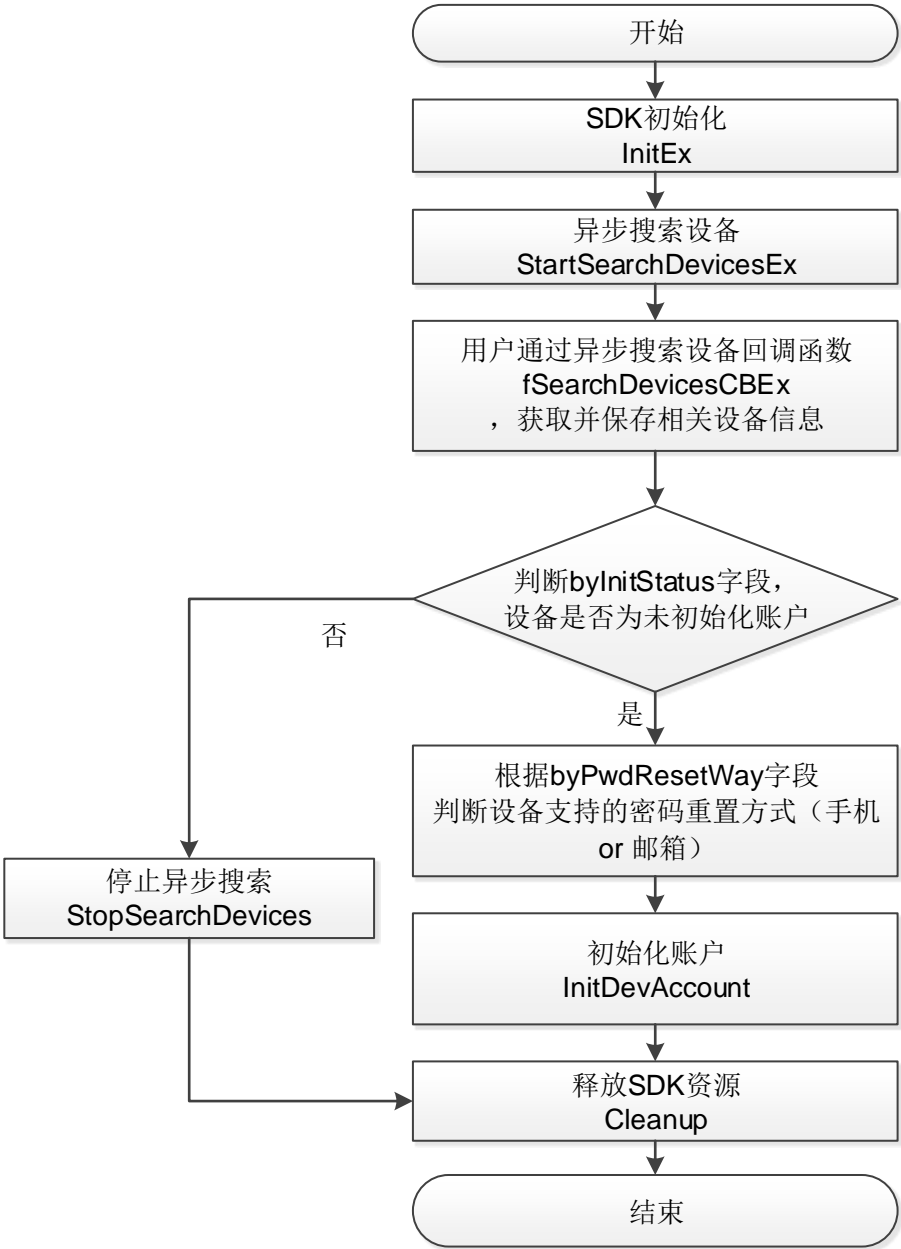
表2-2 设备搜索及初始化账户接口说明

接口	接口说明
InitEx	NetSDK 初始化接口
Cleanup	NetSDK 清理接口
StartSearchDevicesEx	异步搜索同网段设备
StopSearchDevices	停止异步搜索同网段内 IPC、NVS 等设备
SearchDevicesByIPs	跨网段搜索设备
InitDevAccount	初始化设备账户
GetLastError	获取接口调用失败时的错误码接口

## 2.2.3 流程说明

### 2.2.3.1 异步搜索同网段设备

图2-2 异步搜索设备及初始化设备账户流程



### 注意事项

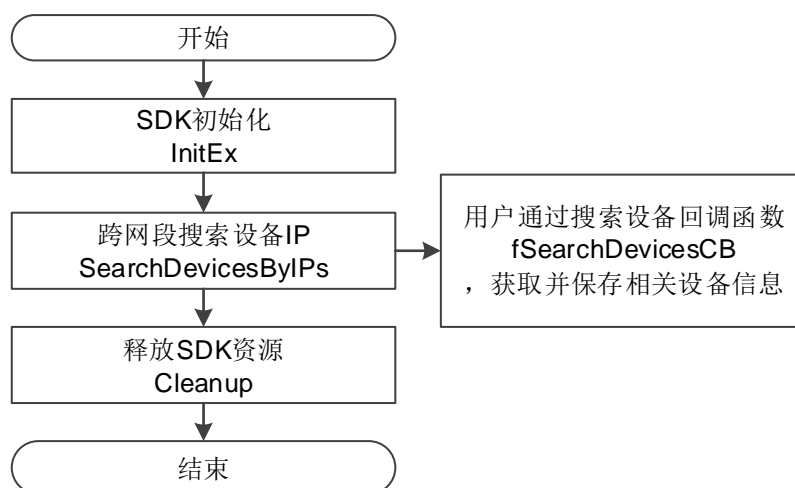
InitEx 只需要在整个 Demo 的开始，使用 NetSDK 接口前调用一次，Cleanup 只需要在所有与 NetSDK 相关功能都使用完毕后调用一次释放 NetSDK 资源即可。这两个接口不需要每用一个功能都调用一次。

## 流程说明

- 步骤1 通过调用 `InitEx` 完成 NetSDK 初始化流程。
- 步骤2 调用 `StartSearchDevicesEx` 搜索设备。
- 步骤3 在搜索回调函数 `fSearchDevicesCBEx` 中匹配需要初始化的设备，根据 `byInitStatus` 字段判断设备为未初始化设备账户，根据 `byPwdResetWay` 字段判断设备支持的密码重置方式，调初始化接口时需要该字段。
- 步骤4 调用 `InitDevAccount` 接口初始化设备账户。
- 步骤5 搜索功能使用完毕后，调用 `StopSearchDevices` 接口停止搜索。
- 步骤6 NetSDK 功能使用完后，调用 `Cleanup` 释放 NetSDK 资源。

### 2.2.3.2 同步搜索跨网段设备

图2-3 同步搜索跨网段设备流程



## 注意事项

`InitEx` 只需要在整个 Demo 的开始，使用 NetSDK 接口前调用一次，`Cleanup` 只需要在所有与 NetSDK 相关功能都使用完毕后调用一次释放 NetSDK 资源即可。这两个接口不需要每用一个功能都调用一次。

## 流程说明

- 步骤1 通过调用 `InitEx` 完成 NetSDK 初始化流程。
- 步骤2 调用 `SearchDevicesByIPs` 搜索设备。用户通过搜索回调函数 `fSearchDevicesCB` 可以获取到设备信息。
- 步骤3 NetSDK 功能使用完后，调用 `Cleanup` 释放 NetSDK 资源。

## 2.2.4 示例代码

### 2.2.4.1 异步搜索同网段设备及初始化账户

#### 代码路径

Demo\SearchDeviceDemo\SearchDeviceDemo.py

#### 示例代码

```
# 组播和广播搜索
def start_search_device(self):
    # 获取本地 IP，考虑到多网卡的情况下搜索
    # 有几张网卡，就得调用几次搜索接口
    IPList = self.getIPAdrrs()
    nSuccess = 0
    for i in range(IPList.__len__()):
        startsearch_in = NET_IN_STARTSERACH_DEVICE()
        startsearch_in.dwSize = sizeof(NET_IN_STARTSERACH_DEVICE)
        startsearch_in.emSendType =
EM_SEND_SEARCH_TYPE.MULTICAST_AND_BROADCAST
        startsearch_in.cbSearchDevices = search_device_callback
        startsearch_in.szLocallp = IPList[i].encode()
        startsearch_out = NET_OUT_STARTSERACH_DEVICE()
        startsearch_out.dwSize = sizeof(NET_OUT_STARTSERACH_DEVICE)
        ISearchHandle = self.sdk.StartSearchDevicesEx(startsearch_in, startsearch_out)
        if ISearchHandle != 0:
            nSuccess += 1
            self.ISearchHandle_list.append(ISearchHandle)
    if(IPList.__len__() > 0):
        del IPList
    if(nSuccess > 0):
        return True
    else:
        return False

# 停止搜索,配合 start_search_device 使用
def stop_search_device(self):
    for i in range(self.ISearchHandle_list.__len__()):
        result = self.sdk.StopSearchDevices(self.ISearchHandle_list[i])
    nUpdateNum = 0
    self.ISearchHandle_list.clear()
    self.device_info_list.clear()
    self.device_mac_list.clear()
    self.tableWidget.clear()
```

```

self.row = 0
self.column = 0
device_queue.queue.clear()
if(not device_queue.empty()):
    device_queue.task_done()
self.tableWidget.setHorizontalHeaderLabels(['序号(No.)', '状态(Status)', 'IP 版本(IP
Version)', 'IP 地址(IP Address)', '端口(Port)', '子网掩码(Subnet Mask)', '网关(Gateway)', '物理地
址(Mac Address)', '设备类型(Device Type)', '详细类型(Detail Type)', 'Http(Http)'])
return

def Init_Btn(self):
    # 获取选中行的 ip 和初始化信息
    currentRow = self.tableWidget.currentRow()
    if((len(self.device_info_list) == 0) or ((self.device_info_list[currentRow][0] & 3) != 1)):
        QMessageBox.about(self, '提示(prompt)', "请选择未初始化设备(Please select not
initialized device)")
    else:
        result = self.init_device_accout(self.device_info_list[currentRow])
        if result == True:
            QMessageBox.about(self, '提示(prompt)', "初始化成功(Initialize Success)")
            item = QTableWidgetItem("已初始化(Initialize)")
            self.device_info_list[currentRow][0] = 2
            self.tableWidget.setItem(currentRow, 1, item)
            self.tableWidget.update()
            self.tableWidget.viewport().update()

# 初始化账号
def init_device_accout(self, device_info:list):
    child = QDialog()
    child_ui = Ui_InitDevAccount()
    child_ui.setupUi(child)
    if (1 == (device_info[3] & 1)):
        # 手机
        child_ui.way_lineEdit.setText('手机(Phone)')
    elif (1 == (device_info[3] >> 1 & 1)):
        # 邮箱
        child_ui.way_lineEdit.setText('邮箱(Mail)')
    value = child.exec()
    if (value == 0):
        return False
    init_Account_In = NET_IN_INIT_DEVICE_ACCOUNT()
    init_Account_In.dwSize = sizeof(init_Account_In)
    init_Account_In.szMac = device_info[2]
    username = child_ui.username_lineEdit.text()
    password = child_ui.password_lineEdit.text()
    confirm_password = child_ui.confirm_password_lineEdit.text()
    if(password != confirm_password):

```



```

        QMessageBox.about(self, '提示(prompt)', "确认密码不一致，请重新输入(Confirm password is wrong, please input again)")
        return
    init_Account_In.szUserName = username.encode()
    init_Account_In.szPwd = password.encode()
    init_Account_In.szCellPhone = child_ui.reset_way_lineEdit.text().encode()
    if (1 == (device_info[3] & 1)):
        # 手机
        init_Account_In.szCellPhone = child_ui.reset_way_lineEdit.text().encode()
    elif(1 == (device_info[3] >> 1 & 1)):
        # 邮箱
        init_Account_In.szMail = child_ui.reset_way_lineEdit.text().encode()
    init_Account_In.byPwdResetWay = device_info[3]
    init_Account_Out = NET_OUT_INIT_DEVICE_ACCOUNT()
    init_Account_Out.dwSize = sizeof(init_Account_Out)

    result = self.sdk.InitDevAccount(init_Account_In, init_Account_Out, 5000,
device_info[4])
    if result:
        return True
    else:
        QMessageBox.about(self, '提示(prompt)', 'error:' + str(self.sdk.GetLastError()))
        return False

```

## 2.2.4.2 同步搜索跨网段设备

### 代码路径

Demo\SearchDeviceDemo\SearchDeviceDemo.py

### 示例代码

```

# 单播搜索
def start_search_device_byIP(self, start_IP, end_IP): #这里要注意每个 ip 的有效性
    startsearchbyIp_in = DEVICE_IP_SEARCH_INFO()
    startsearchbyIp_in.dwSize = sizeof(DEVICE_IP_SEARCH_INFO)
    start = struct.unpack("!!", socket.inet_aton(start_IP))[0] # 网络序转字节序
    end = struct.unpack("!!", socket.inet_aton(end_IP))[0]
    if (end - start > 255):
        QMessageBox.about(self, '提示(prompt)', "IP 数量超过最大限制 256(Number of IP addresses exceeds the upper limit 256.)")
        return False

    startsearchbyIp_in.nIpNum = end - start + 1

    for i in range(startsearchbyIp_in.nIpNum):

```

```

        ip = DEVICE_IP_SEARCH_INFO_IP()
        ip.IP = socket.inet_ntoa(struct.pack("!!", start + i)).encode()
        startsearchbyIp_in.szIP[i] = ip

    wait_time = int(wnd.Searchtime_lineEdit.text())
    # 获取本地 IP，考虑到多网卡的情况下搜索
    # 有几张网卡，就得调用几次搜索接口
    IPList = self.getIPAddrs()
    nSuccessNum = 0
    for i in range(IPList.__len__()):
        result = self.sdk.SearchDevicesByIPs(startsearchbyIp_in,
        search_devie_byIp_callback, 0, IPList[i].encode(), wait_time)
        if result:
            nSuccessNum += 1
    if (IPList.__len__() > 0):
        del IPList
    if(nSuccessNum > 0):
        return True
    else:
        return False

```

## 2.3 设备登录

### 2.3.1 简介

设备登录，即用户鉴权，是进行其他业务的前提。

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

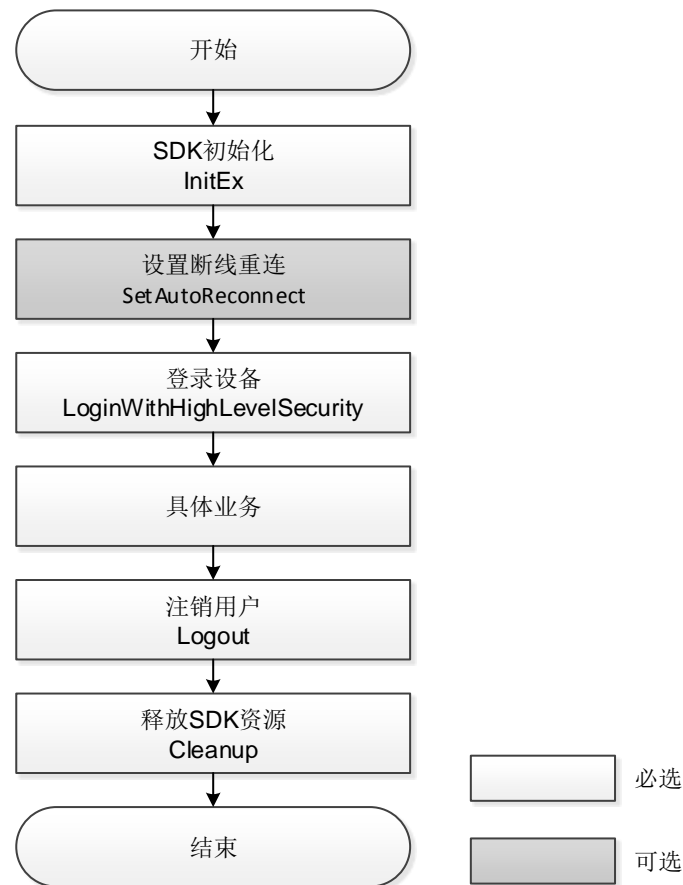
### 2.3.2 接口总览

表2-3 设备登录接口说明

接口	接口说明
InitEx	NetSDK 初始化接口
SetAutoReconnect	设置断线重连接口
Cleanup	NetSDK 清理接口
LoginWithHighLevelSecurity	高安全级别登录接口
Logout	登出接口

### 2.3.3 流程说明

图2-4 登录设备流程图



### 注意事项

InitEx 只需要在整个 Demo 的开始，使用 NetSDK 接口前调用一次，Cleanup 只需要在所有与 NetSDK 相关功能都使用完毕后调用一次释放 NetSDK 资源即可。这两个接口不需要每用一个功能都调用一次。

### 流程说明

- 步骤1 调用 InitEx，完成 NetSDK 初始化流程。
- 步骤2 调用 SetAutoReconnect，设置断线重连函数。
- 步骤3 调用 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 Logout 登出设备。
- 步骤6 NetSDK 功能使用完后，调用 Cleanup 释放 NetSDK 资源。

### 2.3.4 示例代码

```
# 登录设备，获取登录句柄和设备信息，如果失败则显示错误信息
stuInParam = NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY()
stuInParam.dwSize = sizeof(NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY)
```

```

stuInParam.szIP = ip.encode()
stuInParam.nPort = port
stuInParam.szUserName = username.encode()
stuInParam.szPassword = password.encode()
stuInParam.emSpecCap = EM_LOGIN_SPAC_CAP_TYPE.TCP
stuInParam.pCapParam = None

stuOutParam = NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY()
stuOutParam.dwSize = sizeof(NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY)

self.loginID, device_info, error_msg = self.sdk.LoginWithHighLevelSecurity(stuInParam,
stuOutParam)    if self.loginID != 0:
    for i in range(int(device_info.nChanNum)):
        self.Channel_comboBox.addItem(str(i)) # 显示设备的各个通道
    else:
        QMessageBox.critical(self, '提示(prompt)', error_msg, QMessageBox.Ok,
QMessageBox.No) # 显示登录接口的错误信息

# 登出设备
result = self.sdk.Logout(self.loginID)
    if result:
        self.loginID = 0

```

## 2.3.5 注意事项

- 登录句柄：登录成功时接口返回值非 0（即句柄小于 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它各种业务。
- 句柄重复：登录句柄有可能与存在过的句柄相同，属于正常现象。例如登录设备 A 获得 loginIDA，将 loginIDA 注销，再次进行登录操作，可能又获取到 LoginIDA。但是在句柄的整个生命周期内，不会出现重复的句柄。
- 登出：接口内部会释放登录会话中已打开的业务，但建议用户不要依赖登出接口的清理功能。例如打开监视后，在不需要使用监视时，用户应该调用结束监视的接口。
- 登录与登出配对使用，登录会消耗一定的内存和 socket 信息，在登出后释放资源。
- 登录失败：建议通过登录接口的返回值 error\_msg 参数来获取错误信息。具体的错误信息可参考 LoginWithHighLevelSecurity 方法内的错误码列表。
- 设备断线以后，设备的登录 ID 会失效，待设备自动重连成功后该登录 ID 会重新生效。

## 2.4 实时监视

### 2.4.1 简介

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

NetSDK 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，NetSDK 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 NetSDK 接口实现。

### 2.4.2 接口总览

表2-4 实时监控接口说明

接口	说明
InitEx	NetSDK 初始化接口
Cleanup	NetSDK 清理接口
LoginWithHighLevelSecurity	高安全级别登录接口
Logout	登出接口
RealPlayEx	开始实时监控扩展接口
StopRealPlayEx	停止实时监控扩展接口
GetLastError	获取调用 NetSDK 接口失败的错误码接口
GetLastErrorMessage	获取调用 NetSDK 接口失败的错误信息接口

### 2.4.3 流程说明

图2-5 实时监控流程



## 注意事项

InitEx 只需要在整个 Demo 的开始，使用 NetSDK 接口前调用一次，Cleanup 只需要在所有与 NetSDK 相关功能都使用完毕后调用一次释放 NetSDK 资源即可。这两个接口不需要每用一个功能都调用一次。

## 流程说明

- 步骤1 调用 InitEx 完成 NetSDK 初始化流程。
- 步骤2 调用 LoginWithHighLevelSecurity 登录设备。
- 步骤3 调用 RealPlayEx 启动实时监视。
- 步骤4 实时监视使用完毕后，调用 StopRealPlayEx 停止实时监视。
- 步骤5 业务使用完后，调用 Logout 登出设备。
- 步骤6 NetSDK 功能使用完后，调用 Cleanup 释放 NetSDK 资源。

### 2.4.4 示例代码

```
# 开始实时监视
channel = self.Channel_comboBox.currentIndex() # 通道号
if self.StreamTyp_comboBox.currentIndex() == 0:
    stream_type = SDK_RealPlayType.Realplay # 主码流
else:
    stream_type = SDK_RealPlayType.Realplay_1 # 辅码流
self.playID = self.sdk.RealPlayEx(self.loginID, channel, self.PlayWnd.winId(), stream_type)
if self.playID != 0:
    self.play_btn.setText("停止(Stop)")
    self.StreamTyp_comboBox.setEnabled(False)
else:
    QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessage(),
    QMessageBox.No)

# 结束实时监视
result = self.sdk.StopRealPlayEx(self.playID)
if result:
    self.playID = 0
    self.PlayWnd.repaint()
```

### 2.4.5 注意事项

- GetLastError 是调用 NetSDK 的接口失败时，用来获取错误码的接口。GetLastErrorMessage 是获取错误信息的接口。
- 用户使用过程中如果遇到错误，推荐使用 GetLastErrorMessage 接口来获取错误信息来定位错误原因。

## 2.5 录像回放

### 2.5.1 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。

### 2.5.2 接口总览

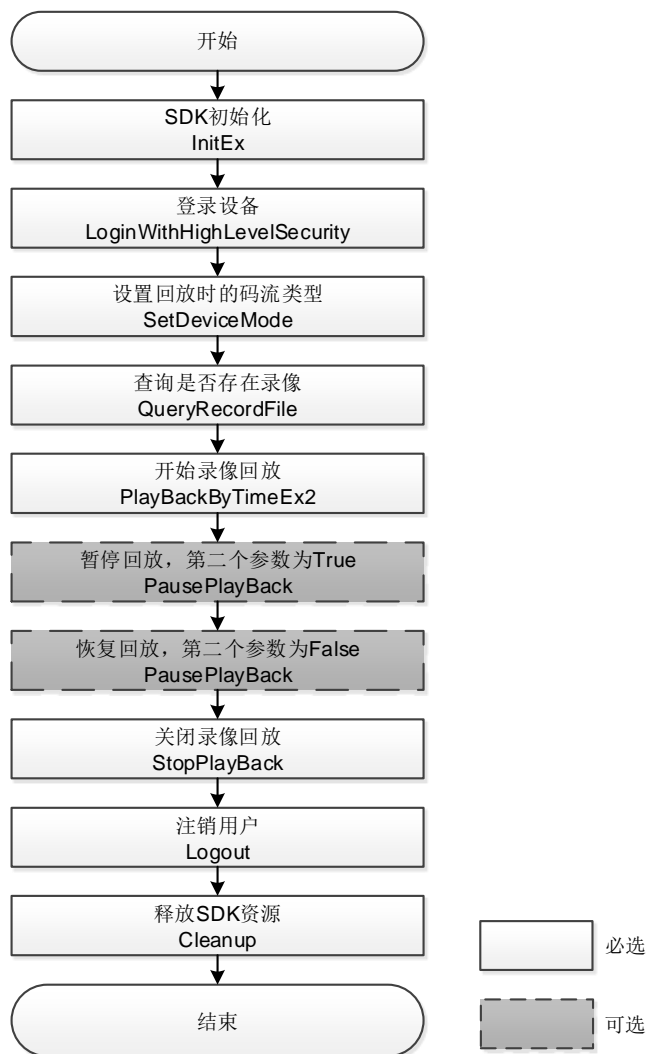
表2-5 录像回放接口说明

接口	说明
InitEx	NetSDK 初始化接口
Cleanup	NetSDK 清理接口
LoginWithHighLevelSecurity	高安全级别登录接口
Logout	登出接口
PlayBackByTimeEx2	按时间方式回放扩展接口
StopPlayBack	停止回放接口
PausePlayBack	暂停或停止暂停接口
SetDeviceMode	设置设备模式接口
QueryRecordFile	查询时间段内的所有录像文件接口

### 2.5.3 流程说明

NetSDK 初始化之后，登录设备，输入通道号、码流类型、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。

图2-6 NetSDK 解码回放流程图



## 流程说明

- 步骤1 调用 InitEx 完成 NetSDK 初始化流程。
- 步骤2 初始化成功后，调用 LoginWithHighLevelSecurity 登录设备。
- 步骤3 调用 SetDeviceMode，设置回放时的码流类型。
- 步骤4 调用 QueryRecordFile，查询选中时间段内是否存在录像。
- 步骤5 调用 PlayBackByTimeEx2 启动回放。
- 步骤6 （可选）调用 PausePlayBack，第二个参数为 True 时暂停回放。
- 步骤7 （可选）调用 PausePlayBack，第二个参数为 False 时恢复回放。
- 步骤8 回放使用完后，调用 StopPlayBack 停止回放。
- 步骤9 业务使用完后，调用 Logout 登出设备。
- 步骤10 NetSDK 功能使用完后，调用 Cleanup 释放 NetSDK 资源。

## 2.5.4 示例代码

```
# 设置回放时的码流类型，此处设置成主码流
stream_type = c_int(0)
result = self.sdk.SetDeviceMode(self.loginID,
```



```

int(EM_USEDEV_MODE.RECORD_STREAM_TYPE), stream_type)
if not result:
    QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessage(),
    QMessageBox.No)

# query record file 查询录像文件
result, fileCount, infos = self.sdk.QueryRecordFile(self.loginID, 0,
int(EM_QUERY_RECORD_TYPE.ALL), startTime, endTime, None, 5000, False)

# 开启录像回放
inParam = NET_IN_PLAY_BACK_BY_TIME_INFO()
inParam.hWnd = self.PlayBackWnd.winId()
inParam.cbDownloadPos = DownloadPosCallBack
inParam.dwPosUser = 0
inParam.fDownloadDataCallBack = DownloadDataCallBack
inParam.dwDataUser = 0
inParam.nPlayDirection = 0
inParam.nWaittime = 5000
inParam.stStartTime.dwYear = start_time.dwYear
inParam.stStartTime.dwMonth = start_time.dwMonth
inParam.stStartTime.dwDay = start_time.dwDay
inParam.stStartTime.dwHour = start_time.dwHour
inParam.stStartTime.dwMinute = start_time.dwMinute
inParam.stStartTime.dwSecond = start_time.dwSecond
inParam.stStopTime.dwYear = end_time.dwYear
inParam.stStopTime.dwMonth = end_time.dwMonth
inParam.stStopTime.dwDay = end_time.dwDay
inParam.stStopTime.dwHour = end_time.dwHour
inParam.stStopTime.dwMinute = end_time.dwMinute
inParam.stStopTime.dwSecond = end_time.dwSecond
outParam = NET_OUT_PLAY_BACK_BY_TIME_INFO()

nchannel = self.Channel_comboBox.currentIndex()
self.playbackID = self.sdk.PlayBackByTimeEx2(self.loginID, nchannel, inParam, outParam)
if self.playbackID != 0:
    self.PlayBack_pushbutton.setText("停止(Stop)")
    self.Pause_pushbutton.setEnabled(True)
    self.Channel_comboBox.setEnabled(False)
    self.StreamTyp_comboBox.setEnabled(False)
    self.Channel_comboBox.repaint()
    self.StreamTyp_comboBox.repaint()
    self.PlayBackWnd.repaint()
else:
    QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessage(),
    QMessageBox.No)

# 暂停录像回放

```

```

result = self.sdk.PausePlayBack(self.playbackID, True)

# 恢复录像回放
result = self.sdk.PausePlayBack(self.playbackID, False)

# 停止回放
result = self.sdk.StopPlayBack(self.playbackID)
if result:
    self.playbackID = 0
if not result:
    QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessage(),
    QMessageBox.No)
return

```

## 2.6 录像下载

### 2.6.1 简介

视频监控系统在平安城市、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 **NetSDK** 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。下载到的录像文件是大华私有格式的录像文件，需要用大华播放器或集成大华 **playsdk** 才能播放。

### 2.6.2 接口总览

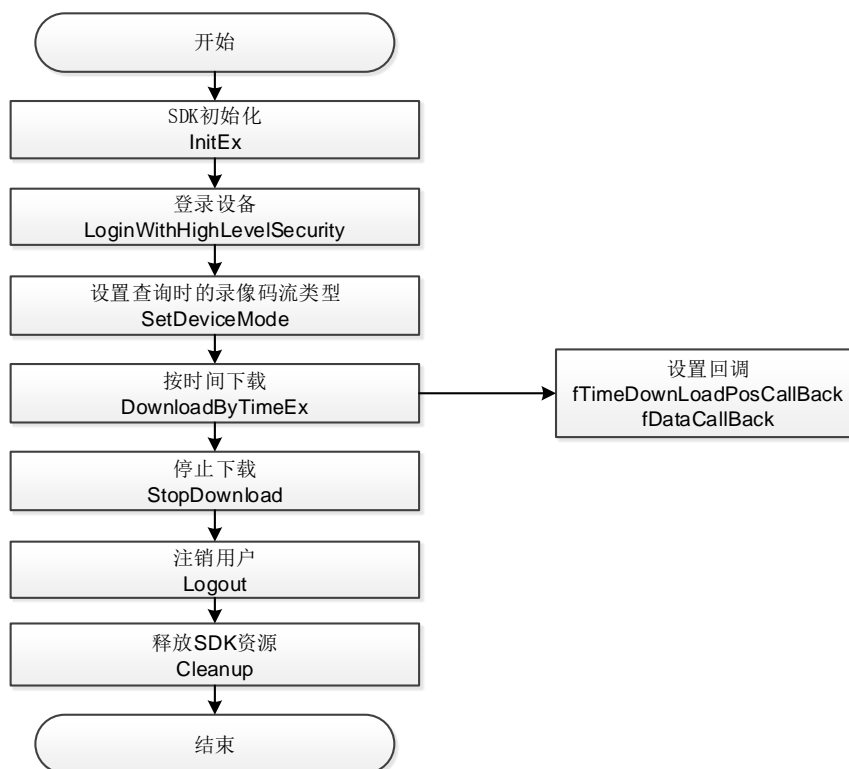
表2-6 录像下载接口说明

接口	说明
InitEx	NetSDK 初始化接口
Cleanup	NetSDK 清理接口
LoginWithHighLevelSecurity	高安全级别登录接口
Logout	登出接口
SetDeviceMode	设置设备模式接口
DownloadByTimeEx	通过时间下载录像接口
StopDownload	停止录像下载

### 2.6.3 流程说明

用户传入需要下载的起始时间和结束时间，**NetSDK** 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，**NetSDK** 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。

图2-7 NetSDK 按时间下载流程图



## 流程说明

- 步骤1 调用 InitEx 完成 NetSDK 初始化流程。
- 步骤2 初始化成功后，调用 LoginWithHighLevelSecurity 登录设备。
- 步骤3 调用 SetDeviceMode，设置下载的码流类型。
- 步骤4 调用 DownloadByTimeEx 启动开始按时间下载，直到下载完成。
- 步骤5 调用 StopDownload 停止下载。
- 步骤6 （可选）在 fTimeDownLoadPosCallBack 回调函数内更新下载进度。
- 步骤7 业务使用完后，调用 Logout 登出设备。
- 步骤8 NetSDK 功能使用完后，调用 Cleanup 释放 NetSDK 资源。

## 2.6.4 示例代码

```

# 设置下载的码流类型，此处设置成主码流
stream_type = c_int(0)
result = self.sdk.SetDeviceMode(self.loginID,
int(EM_USEDEV_MODE.RECORD_STREAM_TYPE), stream_type)
if not result:
    QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessage(),
QMessageBox.No)

# 开启录像下载
start_date = self.Start_dateTimeEdit.date()
start_time = self.Start_dateTimeEdit.time()
startDateTime = NET_TIME()
  
```

```

startDateTime.dwYear = start_date.year()
startDateTime.dwMonth = start_date.month()
startDateTime.dwDay = start_date.day()
startDateTime.dwHour = start_time.hour()
startDateTime.dwMinute = start_time.minute()
startDateTime.dwSecond = start_time.second()

end_date = self.End_dateTimeEdit.date()
end_time = self.End_dateTimeEdit.time()
enddateTime = NET_TIME()
enddateTime.dwYear = end_date.year()
enddateTime.dwMonth = end_date.month()
enddateTime.dwDay = end_date.day()
enddateTime.dwHour = end_time.hour()
enddateTime.dwMinute = end_time.minute()
enddateTime.dwSecond = end_time.second()

save_file_name = 'D:\savedata.dav'# 保存的文件的路径和名称
nchannel = self.Channel_comboBox.currentIndex()
self.downloadID = self.sdk.DownloadByTimeEx(self.loginID, nchannel,
int(EM_QUERY_RECORD_TYPE.ALL), startDateTime, enddateTime, save_file_name,
TimeDownLoadPosCallBack, 0, DownLoadDataCallBack, 0)
if self.downloadID:
    self.Download_pushButton.setText("停止(Stop)")
else:
    QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMassage(),
    QMessageBox.No)

# 停止录像下载
result = self.sdk.StopDownload(self.downloadID)
if result:
    self.downloadID = 0

#回调函数
@WINFUNCTYPE(None, c_longlong, c_ulong, POINTER(c_ubyte), c_ulong, c_longlong)
def DownLoadDataCallBack(IPlayHandle, dwDataType, pBuffer, dwBufSize, dwUser):
    pass

@WINFUNCTYPE(None, c_longlong, c_ulong, c_ulong, c_int,
POINTER(NET_RECORDFILE_INFO), c_ulong)
def TimeDownLoadPosCallBack(IPlayHandle, total_size, download_size, index, recordfileinfo,
dwUser):
    try:
        # 显示进度
        if download_size == 0xffffffff:
            self.downloadID = 0
            self.Download_progressBar.setValue(0)

```

```

        self.sdk.StopDownload(self.downloadID)
        self.Download_pushButton.setText("下载(download)")
        self.Message_label.setText("Download End(下载结束)!")
    elif download_size == 0xffffffff:
        self.downloadID = 0
        self.Download_progressBar.setValue(0)
        self.Download_pushButton.setText("下载(download)")
        self.Message_label.setText("Download Failed(下载失败)!")
    else:
        if download_size >= total_size:
            self.Download_progressBar.setValue(100)
        else:
            percentage = int(download_size * 100 / total_size)
            self.Download_progressBar.setValue(percentage)
except Exception as e:
    print(e)
except Exception as e:
    print(e)

```

## 2.7 设备控制

### 2.7.1 简介

设备控制主要包括设置设备时间、获取设备时间和远程设备重启。

### 2.7.2 接口总览

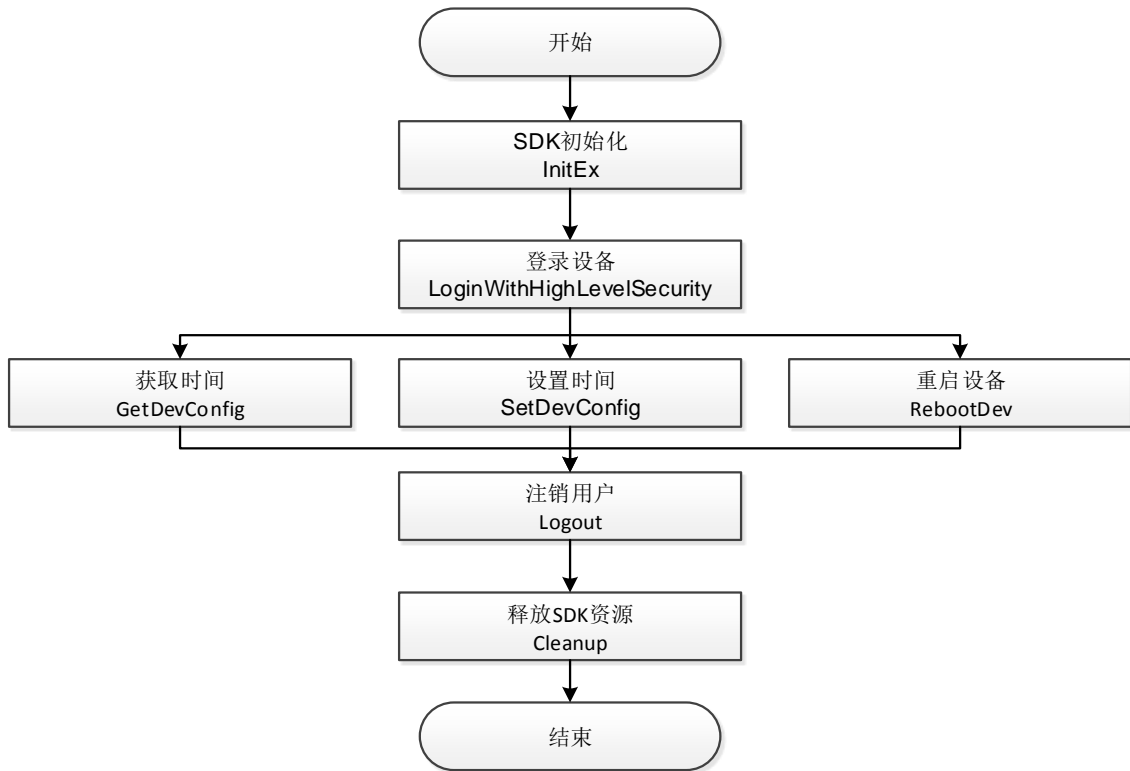
表2-7 设备控制接口说明

接口	说明
InitEx	NetSDK 初始化接口
Cleanup	NetSDK 清理接口
LoginWithHighLevelSecurity	高安全级别登录接口
Logout	登出接口
GetDevConfig	查询配置信息
SetDevConfig	设置配置信息
RebootDev	重启设备

### 2.7.3 流程说明

用户登录设备后便可以直接获取时间或者设置时间或者重启设备。

图2-8 NetSDK 设备控制流程图



## 流程说明

- 步骤1 调用 InitEx 完成 NetSDK 初始化流程。
- 步骤2 初始化成功后，调用 LoginWithHighLevelSecurity 登录设备。
- 步骤3 （可选）调用 GetDevConfig 获取设备时间。
- 步骤4 （可选）调用 SetDevConfig 设置设备时间。
- 步骤5 （可选）调用 RebootDev 重启设备
- 步骤6 业务使用完后，调用 Logout 登出设备。
- 步骤7 NetSDK 功能使用完后，调用 Cleanup 释放 NetSDK 资源。

## 2.7.4 示例代码

```

# 获取设备时间
time = NET_TIME()
result = self.sdk.GetDevConfig(self.loginID, int(EM_DEV_CFG_TYPE.TIMECFG), -1, time,
sizeof(NET_TIME))
if not result:
    QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessage(),
QMessageBox.Ok, QMessageBox.No)
else:
    get_time = QDateTime(time.dwYear, time.dwMonth, time.dwDay, time.dwHour,
time.dwMinute, time.dwSecond)
    self.Time_dateTimeEdit.setDateTime(get_time)

# 设置设备时间

```

```
device_date = self.Time_dateTimeEdit.date()
device_time = self.Time_dateTimeEdit.time()
deviceDateTime = NET_TIME()
deviceDateTime.dwYear = device_date.year()
deviceDateTime.dwMonth = device_date.month()
deviceDateTime.dwDay = device_date.day()
deviceDateTime.dwHour = device_time.hour()
deviceDateTime.dwMinute = device_time.minute()
deviceDateTime.dwSecond = device_time.second()

result = self.sdk.SetDevConfig(self.loginID, int(EM_DEV_CFG_TYPE.TIMECFG), -1,
deviceDateTime, sizeof(NET_TIME))
if not result:
    QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessage(),
QMessageBox.Ok, QMessageBox.No)

# 重启设备
result = self.sdk.RebootDev(self.loginID)
if not result:
    QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessage(),
QMessageBox.Ok, QMessageBox.No)
```

## 2.8 远程抓图

### 2.8.1 简介

远程抓图指从设备中获取图片，即用户调用 **NetSDK** 接口发送抓图命令给设备，设备在实时监控中抓取当前画面，并发送给 **NetSDK**，**NetSDK** 将接收到的图片数据返回给用户，用于上层用户实现平台开发需求。

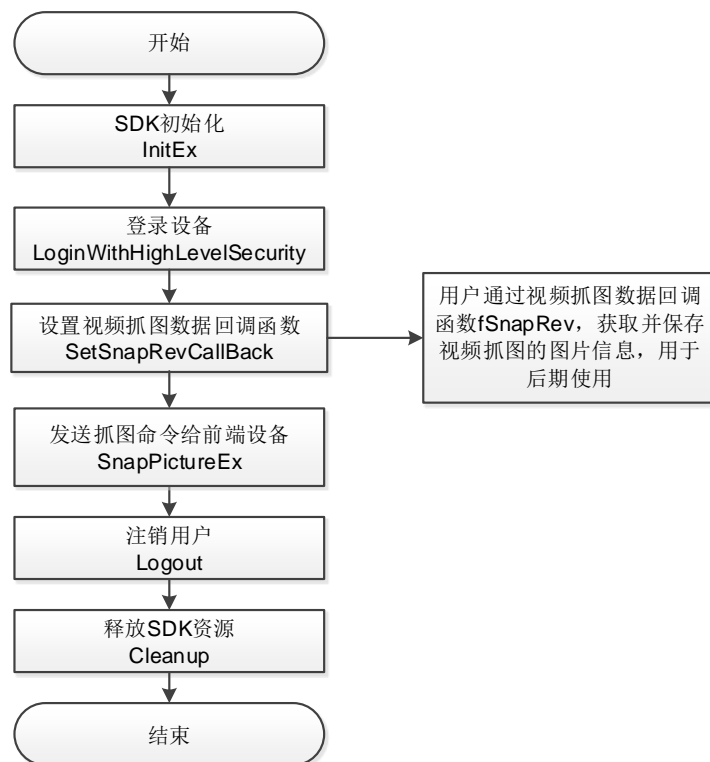
### 2.8.2 接口总览

表2-8 远程抓图接口说明

接口	接口说明
InitEx	NetSDK 初始化接口
Cleanup	NetSDK 清理接口
LoginWithHighLevelSecurity	高安全级别登录接口
SetSnapRevCallBack	设置远程抓图数据回调函数
SnapPictureEx	抓图请求扩展接口
Logout	登出接口
GetLastError	获取接口调用失败时的错误码接口

## 2.8.3 流程说明

图2-9 远程抓图流程



### 注意事项

- InitEx 只需要在整个 Demo 的开始，使用 NetSDK 接口前调用一次，Cleanup 只需要在所有与 NetSDK 相关功能都使用完毕后调用一次释放 NetSDK 资源即可。这两个接口不需要每用一个功能都调用一次。
- 抓图时间间隔设置为 1 秒以上，建议设置为 3 秒。

### 流程说明

- 步骤1 通过调用 InitEx 完成 NetSDK 初始化流程。
- 步骤2 初始化成功后，调用 LoginWithHighLevelSecurity 登录设备。
- 步骤3 调用 SetSnapRevCallBack 设置抓图回调函数，当 NetSDK 收到设备端发送过来的抓图数据时，会调用 fSnapRev 回调函数回调图片信息及图片数据给用户。
- 步骤4 调用 SnapPictureEx 发送抓图命令给前端设备，在 fSnapRev 回调函数中等待设备回复的图片信息。
- 步骤5 调用 Logout，注销用户。
- 步骤6 NetSDK 功能使用完后，调用 Cleanup 释放 NetSDK 资源。



## 2.8.4 示例代码

### 代码路径

Demo\CapturePicture\CaptureDemo.py

### 示例代码

```
def capture_btn_onclick(self):
    # 设置抓图回调
    dwUser = 0
    self.sdk.SetSnapRevCallBack(CaptureCallBack, dwUser)
    channel = self.Channel_comboBox.currentIndex()
    snap = SNAP_PARAMS()
    snap.Channel = channel
    snap.Quality = 1
    snap.mode = 0
    # 抓图
    self.sdk.SnapPictureEx(self.loginID, snap)
```

## 2.9 报警上报

### 2.9.1 简介

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

报警上报实现方式为 NetSDK 主动连接设备，并向设备订阅报警功能，设备检测到报警事件立即发送给 NetSDK。

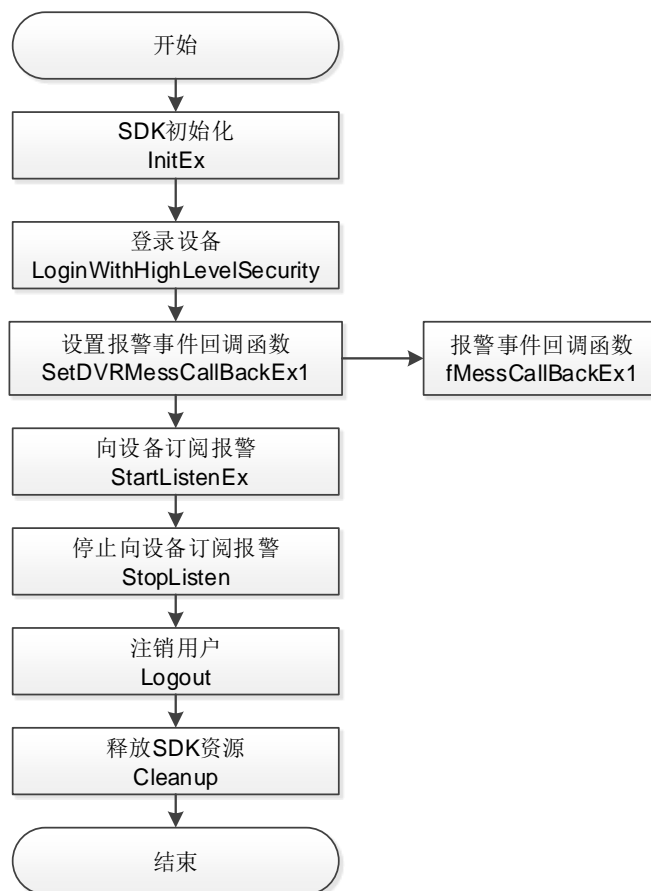
### 2.9.2 接口总览

表2-9 报警上报接口说明

接口	接口说明
InitEx	NetSDK 初始化接口
Cleanup	NetSDK 清理接口
LoginWithHighLevelSecurity	高安全级别登录接口
SetDVRMessCallBackEx1	设置报警回调函数接口
StartListenEx	订阅报警扩展接口
StopListen	停止订阅报警
Logout	登出接口
GetLastError	获取接口调用失败时的错误码接口

## 2.9.3 流程说明

图2-10 报警上报流程



### 注意事项

InitEx 只需要在整个 Demo 的开始，使用 NetSDK 接口前调用一次，Cleanup 只需要在所有与 NetSDK 相关功能都使用完毕后调用一次释放 NetSDK 资源即可。这两个接口不需要每用一个功能都调用一次。

### 流程说明

- 步骤1 通过调用 InitEx 完成 NetSDK 初始化流程。
- 步骤2 初始化成功后，调用 LoginWithHighLevelSecurity 登录设备。
- 步骤3 调用 SetDVRMessCallBackEx1 报警事件回调函数，该接口需在报警订阅之前调用。
- 步骤4 调用 StartListenEx，向设备订阅报警。订阅成功后，设备上报的报警事件通过 fMessCallBackEx1 回调函数通知用户。
- 步骤5 报警上报功能使用完毕后，调用 StopListen，停止向设备订阅报警。
- 步骤6 业务使用完后，调用 Logout 登出设备。
- 步骤7 NetSDK 功能使用完后，调用 Cleanup 释放 NetSDK 资源。

## 2.9.4 示例代码

### 代码路径

Demo\AlarmListen\AlarmListenDemo.py

### 示例代码

```
def __init__(self):
    super(StartListenWnd, self).__init__()
    self.setupUi(self)
    # 界面初始化
    self.init_ui()

    # NetSDK 用到的相关变量和回调
    self.loginID = C_LLONG()
    self.m_DisConnectCallBack = fDisConnect(self.DisConnectCallBack)
    self.m_ReConnectCallBack = fHaveReConnect(self.ReConnectCallBack)

    # 获取 NetSDK 对象并初始化
    self.sdk = NetClient()
    self.sdk.InitEx(self.m_DisConnectCallBack)
    self.sdk.SetAutoReconnect(self.m_ReConnectCallBack)
    #设置报警回调函数
    self.sdk.SetDVRMessCallBackEx1(MessCallback,0)

def attach_btn_onclick(self):
    self.row = 0
    self.column = 0
    self.Alarmlisten_tableWidget.clear()
    self.Alarmlisten_tableWidget.setHorizontalHeaderLabels(['序号(No.)', '时间 (Time)', '通道
(Channel)', '报警类型(Alarm Type)', '状态(Status)'])
    result = self.sdk.StartListenEx(self.loginID)
    if result:
        QMessageBox.about(self, '提示(prompt)', "报警监听成功(Subscribe alarm success)")
        self.Stopalarmlisten_pushButton.setEnabled(True)
        self.Alarmlisten_pushButton.setEnabled(False)
    else:
        QMessageBox.about(self, '提示(prompt)', 'error:' + str(self.sdk.GetLastError()))

def detach_btn_onclick(self):
    if (self.loginID > 0):
        self.sdk.StopListen(self.loginID)
        self.Stopalarmlisten_pushButton.setEnabled(False)
        self.Alarmlisten_pushButton.setEnabled(True)
```

## 2.10 智能事件上报

### 2.10.1 简介

智能事件上报，是指设备端通过对实时监视码流的智能分析，根据用户在设备端配置的智能事件触发规则来判断是否需要上报事件以及是否携带图片给用户。智能事件包括但不限于人脸识别、人脸检测、交通卡口、穿越警戒线、进入警戒区、离开警戒区、在警戒区内、穿越围栏、徘徊检测、遗留检测、搬移检测、物品保护、非法停车、快速移动、逆行检测等。

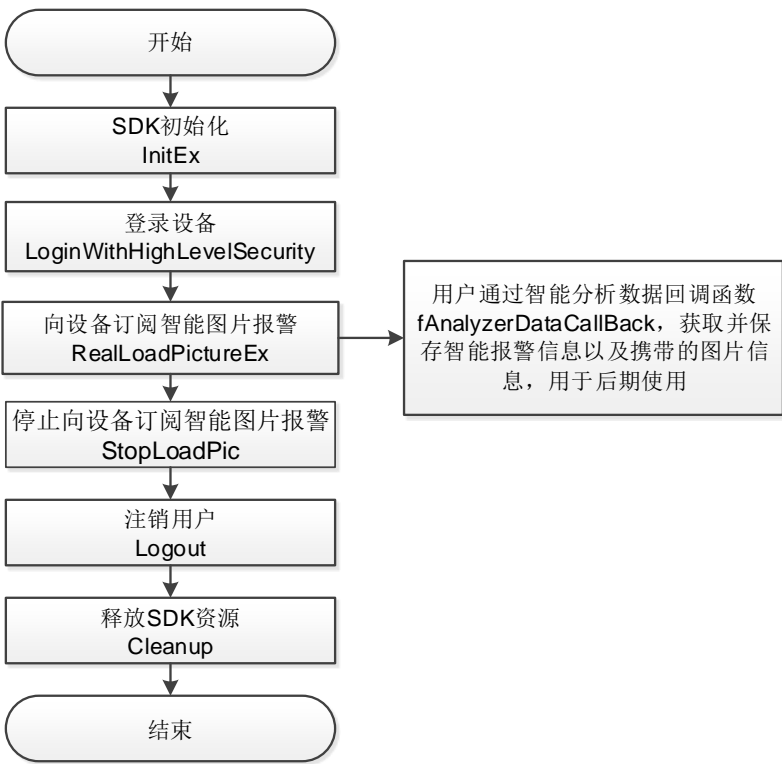
### 2.10.2 接口总览

表2-10 智能事件上报接口说明

接口	接口说明
InitEx	NetSDK 初始化接口
Cleanup	NetSDK 清理接口
LoginWithHighLevelSecurity	高安全级别登录接口
RealLoadPictureEx	智能图片报警订阅接口
StopLoadPic	停止上传智能分析数据-图片
Logout	登出接口
GetLastError	获取接口调用失败时的错误码接口

### 2.10.3 流程说明

图2-11 智能事件上报流程



## 注意事项

**InitEx** 只需要在整个 **Demo** 的开始，使用 **NetSDK** 接口前调用一次，**Cleanup** 只需要在所有与 **NetSDK** 相关功能都使用完毕后调用一次释放 **NetSDK** 资源即可。这两个接口不需要每用一个功能都调用一次。

## 流程说明

- 步骤1 通过调用 **InitEx** 完成 **NetSDK** 初始化流程。
- 步骤2 初始化成功后，调用 **LoginWithHighLevelSecurity** 登录设备。
- 步骤3 调用 **RealLoadPictureEx**, 类型 **dwAlarmType** 对应的 **EM\_EVENT\_IVS\_TYPE** 枚举值，向设备订阅智能抓图报警。订阅成功后，设备上报的智能抓图报警事件通过 **fAnalyzerDataCallBack** 设置的回调函数通知用户，回调函数的主要任务是显示事件和保存事件。
- 步骤4 智能图片报警上报功能使用完毕后，调用 **StopLoadPic** 停止向设备订阅智能图片报警。
- 步骤5 业务使用完后，调用 **Logout** 登出设备。
- 步骤6 **NetSDK** 功能使用完后，调用 **Cleanup** 释放 **NetSDK** 资源。

## 2.10.4 示例代码

### 2.10.4.1 智能交通卡口

#### 代码路径

\\Demo\\IntelligentTrafficDemo

#### 示例代码

```
# 智能交通卡口事件订阅
def attach_btn_onclick(self):
    self.Attach_tableWidget.setHorizontalHeaderLabels(['时间(Time)', '事件(Event)', '车牌号(Plate No.)', '车牌颜色(Plate Color)', '车型类型(Vehicle Type)', '车身颜色(Vehicle Color)'])
    channel = self.Channel_comboBox.currentIndex()
    bNeedPicFile = 1
    dwUser = 0
    self.attachID = self.sdk.RealLoadPictureEx(self.loginID, channel, EM_EVENT_IVS_TYPE.TRAFFICJUNCTION, bNeedPicFile, AnalyzerDataCallBack, dwUser, None)
    if not self.attachID:
        QMessageBox.about(self, '提示(prompt)', 'error:' + str(self.sdk.GetLastError()))
    else:
        self.Attach_pushButton.setEnabled(False)
        self.Detach_pushButton.setEnabled(True)
        QMessageBox.about(self, '提示(prompt)', "订阅成功(Subscribe success)")
```

```
# 取消订阅
def detach_btn_onclick(self):
    if (self.attachID == 0):
        return
    self.sdk.StopLoadPic(self.attachID)
    self.attachID = 0
    self.Attach_pushButton.setEnabled(True)
    self.Detach_pushButton.setEnabled(False)
    self.Attach_tableWidget.clear()
    self.row = 0
    self.column = 0
    self.Attach_tableWidget.viewport().update()
    self.Attach_tableWidget.setHorizontalHeaderLabels(['时间(Time)', '事件(Event)', '车牌号(Plate No.)', '车牌颜色(Plate Color)', '车型类型(Vehicle Type)', '车身颜色(Vehicle Color)'])
```

## 2.10.4.2 人脸识别事件

### 代码路径

Demo\FaceRecognitionDemo\FaceRecognitionDemo.py

### 示例代码

```
def listenevent_btn_onclick(self):
    if not self.realloaID:
        channel = self.Channel_comboBox.currentIndex()
        self.realloaID = self.sdk.RealLoadPictureEx(self.loginID, channel,
EM_EVENT_IVS_TYPE.ALL, True, self.m_AnalyzerDataCallBack)
        if self.realloaID != 0:
            self.ListenEvent_pushButton.setText("取消订阅(Detach Listen)")
        else:
            QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessag(),
QMessageBox.No)
    else:
        result = self.sdk.StopLoadPic(self.realloaID)
        if result:
            self.ListenEvent_pushButton.setText("订阅事件(Listen Event)")
            self.realloaID = 0
        else:
            QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessag(),
QMessageBox.No)

def AnalyzerDataCallBack(self, IAnalyzerHandle, dwAlarmType, pAlarmInfo, pBuffer,
dwBufSize, dwUser, nSequence, reserved):
    if IAnalyzerHandle == self.realloaID:
```

```

        if dwAlarmType == EM_EVENT_IVS_TYPE.FACERECOGNITION:
            alarm_info = cast(pAlarmInfo,
POINTER(DEV_EVENT_FACERECOGNITION_INFO)).contents
            self.show_recognition_info(alarm_info, pBuffer, dwBufSize)

```

### 2.10.4.3 人脸检测事件

#### 代码路径

Demo\FaceRecognitionDemo\FaceRecognitionDemo.py

#### 示例代码

```

def listenevent_btn_onclick(self):
    if not self.realloaID:
        channel = self.Channel_comboBox.currentIndex()
        self.realloaID = self.sdk.RealLoadPictureEx(self.loginID, channel,
EM_EVENT_IVS_TYPE.ALL, True, self.m_AnalyzerDataCallBack)
        if self.realloaID != 0:
            self.ListenEvent_pushButton.setText("取消订阅(Detach Listen)")
        else:
            QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessage(),
QMessageBox.No)
    else:
        result = self.sdk.StopLoadPic(self.realloaID)
        if result:
            self.ListenEvent_pushButton.setText("订阅事件(Listen Event)")
            self.realloaID = 0
        else:
            QMessageBox.critical(self, '提示(prompt)', self.sdk.GetLastErrorMessage(),
QMessageBox.No)

def AnalyzerDataCallBack(self, IAnalyzerHandle, dwAlarmType, pAlarmInfo, pBuffer,
dwBufSize, dwUser, nSequence, reserved):
    if IAnalyzerHandle == self.realloaID:
        if dwAlarmType == EM_EVENT_IVS_TYPE.FACEDETECT:
            alarm_info = cast(pAlarmInfo,
POINTER(DEV_EVENT_FACEDETECT_INFO)).contents
            self.show_detect_info(alarm_info, pBuffer, dwBufSize)

```

## 第 3 章 接口函数

### 3.1 NetSDK 初始化

#### 3.1.1 NetSDK 初始化 InitEx

表3-1 NetSDK 初始化 InitEx

选项	说明	
描述	对整个 NetSDK 进行初始化。	
函数	<pre>def InitEx(cls, call_back: fDisconnect = None, user_data: C_LDWORD = 0, init_param: NETSDK_INIT_PARAM = NETSDK_INIT_PARAM() ) -&gt; int</pre>	
参数	[in] call_back	断线回调函数。
	[in] user_data	断线回调函数的用户参数。
	[in] init_param	初始化参数。
返回值	成功返回 1，失败返回 0。	
说明	<ul style="list-style-type: none"><li>调用网络 NetSDK 其他函数的前提。</li><li>回调函数设置成 None 时，设备断线后不会回调给用户。</li><li>InitEx 传入的 user_data 参数，会在回调函数 fDisconnect 内的同字段 user_data 中返回，user_data 在 NetSDK 内部不做处理，仅用于把用户的数据携带到回调函数中。</li></ul>	

#### 3.1.2 NetSDK 清理 Cleanup

表3-2 NetSDK 清理 Cleanup

选项	说明
描述	清理 NetSDK。
函数	def Cleanup(cls)
参数	无。
返回值	无。
说明	NetSDK 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 SetAutoReconnect

表3-3 设置断线重连回调函数 SetAutoReconnect

选项	说明
描述	设置自动重连回调函数。



选项	说明	
函数	<pre>def SetAutoReconnect(cls, call_back: fHaveReConnect, user_data: C_LDWORD = None )</pre>	
参数	[in] call_back	断线重连回调函数。
	[in] user_data	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 <b>None</b> ，则不自动重连。	

## 3.2 设备搜索及初始化设备账户

### 3.2.1 异步搜索设备 StartSearchDevicesEx

表3-4 异步搜索设备 StartSearchDevicesEx

选项	说明	
描述	异步搜索设备。	
函数	<pre>def StartSearchDevicesEx(cls, pInBuf: NET_IN_STARTSERACH_DEVICE, pOutBuf: NET_OUT_STARTSERACH_DEVICE ) -&gt; C_LLONG</pre>	
参数	[in] pInBuf	异步搜索设备输入结构体。
	[out] pOutBuf	异步搜索设备输出结构体。
返回值	成功返回搜索句柄，失败返回 0，获取具体错误代码调用 <b>GetLastError</b> 。	
说明	该接口只支持搜索同网段内的设备。PC 调用 <b>StartSearchDevicesEx</b> 的次数取决于网卡的数量，例如有 2 张网卡，则 PC 需要调用 2 次 <b>StartSearchDevicesEx</b> 。搜索设备成功后将搜索句柄和 IP 绑定，等回调搜索结果返回后通过搜索句柄找到对应的本地 IP，在初始化设备账户时传入本地 IP。	

### 3.2.2 跨网段搜索设备 SearchDevicesByIPs

表3-5 跨网段搜索设备 IP

选项	说明	
描述	跨网段搜索设备 IP	
函数	<pre>def SearchDevicesByIPs(cls, plpSearchInfo: DEVICE_IP_SEARCH_INFO, cbSearchDevices: fSearchDevicesCB, dwUserData: C_LDWORD, szLocallp: c_char_p = None, dwWaitTime: C_DWORD = 5000 ) -&gt; c_int:</pre>	
参数	[in] plpSearchInfo	搜索设备信息。
	[in] cbSearchDevices	搜索设备回调函数，当有设备响应包回复过来时，

选项	说明	
		NetSDK 将响应包解析成有效的信息，通过回调函数通知用户，具体参见 fSearchDevicesCB 回调函数说明。 回调函数不可为空。
	[in] dwUserData	用户数据，NetSDK 通过搜索设备回调函数 fSearchDevicesCB 将该数据返回给用户，以便用户后续操作，dwUserData 在 NetSDK 内部不做处理。
	[in] szLocallp	本地 IP。可不输入，默认为 None。
	[in] dwWaitTime	用户期望搜索时间。 用户根据自己需求合理设置该参数，由于该接口为同步接口，只有等待搜索时间到达才会从接口返回。
返回值	成功返回 1，失败返回 0。	
说明	该接口为同步接口，只有等待搜索时间到达才会从接口返回，用户需要根据自身网络情况决定搜索时间。	

### 3.2.3 停止异步搜索同网段内设备 StopSearchDevices

表3-6 停止异步搜索同网段内 IPC、NVS 等设备 StopSearchDevices

选项	说明	
描述	停止异步搜索同网段内 IPC、NVS 等设备。	
函数	def StopSearchDevices(cls, ISearchHandle: C_LLONG ) -> c_int	
参数	[in] ISearchHandle	异步搜索设备 ID，对应 StartSearchDevicesEx 等异步搜索设备接口的返回值。
返回值	成功返回 1，失败返回 0。	
说明	该接口与 StartSearchDevicesEx 接口配对使用。	

### 3.2.4 初始化设备账户 InitDevAccount

表3-7 初始化设备账户 InitDevAccount

选项	说明	
描述	初始化设备账户。	
函数	def InitDevAccount(cls, pInitAccountIn: NET_IN_INIT_DEVICE_ACCOUNT, pInitAccountOut: NET_OUT_INIT_DEVICE_ACCOUNT, dwWaitTime: int = 5000, szLocallp: c_char_p = None ) -> c_int	
参数	[in] pInitAccountIn	初始化设备账户输入结构体。
	[out] pInitAccountOut	初始化设备账户输出结构体。
	[in] dwWaitTime	等待时间，单位：ms。
	[in] szLocallp	本地 IP，需要和 StartSearchDevicesEx 的 pInBuf

选项	说明	
		的 szLocalIp 字段相同。
返回值	成功返回 1，失败返回 0。	
说明	PC 有几张网卡，就需要调用几次 StartSearchDevicesEx，搜索成功后将搜索句柄和 IP 绑定。在搜索回调信息时，通过搜索句柄找到对应的本地 IP，在初始化的时候，szLocalIp 需要填成本地 IP。	

## 3.3 设备登录

### 3.3.1 用户登录设备 LoginWithHighLevelSecurity

表3-8 用户登录设备 LoginWithHighLevelSecurity

选项	说明	
描述	用户登录设备。	
函数	<pre>def LoginWithHighLevelSecurity(cls,     stuInParam: NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY,     stuOutParam: NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY ) -&gt; tuple</pre>	
参数	[in] stuInParam	传入参数结构体
	[out] stuOutParam	传出参数结构体
	[out]device_info	设备信息
	[out]error_message	登录接口错误信息。
返回值	成功返回非 0 值，失败返回 0	
说明	无。	

### 3.3.2 用户登出设备 Logout

表3-9 用户登出设备 Logout

选项	说明	
描述	用户登出设备。	
函数	<pre>def Logout(cls,     login_id: int ) -&gt; int</pre>	
参数	[in]login_id	LoginWithHighLevelSecurity 的返回值。
返回值	成功返回 1，失败返回 0。	
说明	无。	

### 3.4 实时监视

#### 3.4.1 打开监视 RealPlayEx

表3-10 打开监视 RealPlayEx

选项	说明	
描述	打开实时监视。	
函数	<pre>def RealPlayEx(cls, login_id: int, channel: int, hwnd: int, play_type=SDK_RealPlayType.Realplay ) -&gt; C_LLONG</pre>	
参数	[in] login_id	LoginWithHighLevelSecurity 的返回值。
	[in] channel	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] play_type	预览类型。
返回值	成功返回非 0 值，失败返回 0。	
说明	在 Windows 环境下： <ul style="list-style-type: none"><li>• hwnd 为有效值时，在对应窗口显示画面。</li><li>• hwnd 为 None 时，取流方式为通过设置回调函数来获取视频数据，再交由用户处理。</li></ul>	

预览类型及含义请参见表 3-11。

表3-11 预览类型说明

预览类型	含义
Realplay	实时预览。
Multiplay	多画面预览。
Realplay_0	实时监视-主码流，等同于 Realplay。
Realplay_1	实时监视-从码流 1。
Realplay_2	实时监视-从码流 2。
Realplay_3	实时监视-从码流 3。
Multiplay_1	多画面预览—1 画面。
Multiplay_4	多画面预览—4 画面。
Multiplay_8	多画面预览—8 画面。
Multiplay_9	多画面预览—9 画面。
Multiplay_16	多画面预览—16 画面。
Multiplay_6	多画面预览—6 画面。
Multiplay_12	多画面预览—12 画面。
Multiplay_25	多画面预览—25 画面。
Multiplay_36	多画面预览—36 画面。

### 3.4.2 关闭监视 StopRealPlay

表3-12 关闭监视 StopRealPlay

选项	说明	
描述	关闭实时监视。	
函数	<pre>def StopRealPlayEx(cls,     realplay_id: int     ) -&gt; int</pre>	
参数	[in] realplay_id	RealPlayEx 的返回值。
返回值	成功返回 1，失败返回 0。	
说明	无。	

## 3.5 录像回放

### 3.5.1 设置工作模式 SetDeviceMode

表3-13 设置工作模式 SetDeviceMode

选项	说明	
描述	设置工作模式。	
函数	<pre>def SetDeviceMode(cls,     login_id: int,     emType: int,     value: c_void_p     ) -&gt; c_int</pre>	
参数	[in] login_id	LoginWithHighLevelSecurity 返回值。
	[in] emType	工作模式枚举。
	[in] value	相应工作模式对应的结构体。
返回值	成功返回 1，失败返回 0。	
说明	无。	

表3-14 工作模式枚举及结构体对照表

emType 枚举	含义	结构体
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"><li>0：主辅码流。</li><li>1：主码流。</li><li>2：辅码流。</li></ul>	无。
RECORD_TYPE	设置按时间录像回放及下载的录像文件类型。	EM_RECORD_TYPE

### 3.5.2 查询时间段内的所有录像文件 QueryRecordFile

表3-15 查询时间段内的所有录像文件 QueryRecordFile

选项	说明	
描述	查询时间段内的所有录像文件。	
函数	<pre>def QueryRecordFile(cls, login_id: int, channel_id: int, recordfile_type: int, start_time: NET_TIME, end_time: NET_TIME, card_id: str, wait_time:int, is_querybytime:bool ) -&gt; tuple</pre>	
参数	[in] login_id	LoginWithHighLevelSecurity 返回值。
	[in] channel_id	设备通道号。
	[in] recordfile_type	查询类型，参考 EM_QUERY_RECORD_TYPE。
	[in] start_time	起始时间。
	[in] end_time	结束时间。
	[in] card_id	卡号。
	[in] wait_time	超时时间。
	[in] is_querybytime	是否是按时间查询。
	[out] file_count	返回文件个数。
	[out] recordfile_infos	返回的录像文件信息，是一个 NET_RECORDFILE_INFO 结构数组。
返回值	成功返回 1，失败返回 0。	
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。	

### 3.5.3 按时间方式回放 PlayBackByTimeEx2

表3-16 按时间方式回放 PlayBackByTimeEx2

选项	说明	
描述	按时间方式回放。	
函数	<pre>def PlayBackByTimeEx2(cls, login_id: int, channel_id: int, in_param: NET_IN_PLAY_BACK_BY_TIME_INFO, out_param: NET_OUT_PLAY_BACK_BY_TIME_INFO ) -&gt; int:</pre>	
参数	[in] login_id	LoginWithHighLevelSecurity 返回值。
	[in] channel_id	设备通道号。
	[in] in_param	查询输入条件。

选项	说明	
	[out] out_param	查询输出信息。
返回值	成功返回非 0 值，失败返回 0。	
说明	<ul style="list-style-type: none"> <li>NET_IN_PLAY_BACK_BY_TIME_INFO 中回调函数声明 cbDownloadPos 和 fDownloadDataCallBack，参见“第 4 章回调函数定义”。</li> <li>pstNetIn 参数中的 hWnd 和 fDownloadDataCallBack 不能同时为 None，否则接口调用会返回失败。</li> </ul>	

### 3.5.4 停止录像回放 StopPlayBack

表3-17 停止录像回放 StopPlayBack

选项	说明	
描述	停止录像回放。	
函数	def StopPlayBack(cls, playback_id: int) -> int	
参数	[in] playback_id	回放句柄，PlayBackByTimeEx2 返回值。
返回值	成功返回 1，失败返回 0。	
说明	无。	

### 3.5.5 暂停或继续录像回放 PausePlayBack

表3-18 暂停或继续录像回放 PausePlayBack

选项	说明	
描述	暂停或继续录像回放。	
函数	def PausePlayBack(cls, playback_id: int, is_pause: bool) -> int:	
参数	[in] playback_id	回放句柄，PlayBackByTimeEx2 返回值。
	[in] is_pause	操作动作，暂停还是继续。True：暂停；False：继续。
返回值	成功返回 1，失败返回 0。	
说明	无。	

## 3.6 录像下载

### 3.6.1 按时间下载录像 DownloadByTimeEx

表3-19 按时间下载录像 DownloadByTimeEx

选项	说明
描述	按时间下载录像。

选项	说明	
函数	<pre>def DownloadByTimeEx(cls, login_id: int, channel_id: int, recordfile_type: int, start_time: NET_TIME, end_time: NET_TIME, save_filename: str, callback_timedownloadpos: fTimeDownLoadPosCallBack, time_UserData: C_LDWORD, callback_timedownloaddata: fDataCallBack, data_UserData: C_LDWORD, pReserved: int = 0 ) -&gt; int</pre>	
参数	[in] login_id	LoginWithHighLevelSecurity 返回值。
	[in] channel_id	设备通道号，从 0 开始。
	[in] recordfile_type	录像文件类型。
	[in] start_time	下载起始时间。
	[in] end_time	下载结束时间。
	[in] save_filename	要保存的录像文件名，全路径。
	[in] callback_timedownloadpos	下载进度回调函数。
	[in] time_UserData	下载进度回调用户自定义数据。
	[in] callback_timedownloaddata	下载数据回调函数。
	[in] data_UserData	下载数据回调用户自定义数据。
	[in] pReserved	保留参数
返回值	成功返回非 0 值，失败返回 0。	
说明	<ul style="list-style-type: none"> <li>调函数声明 callback_timedownloadpos 和 callback_timedownloaddata 请参见“4.6 回放数据回调 fDataCallBack”和“4.7 按时间下载进度回调函数 fTimeDownLoadPosCallBack”。</li> <li>save_filename 不为空，录像数据写入到该路径对应的文件。</li> <li>callback_timedownloaddata 不为空，录像数据通过回调函数返回。</li> </ul>	

### 3.6.2 停止录像下载 StopDownload

表3-20 停止录像下载 StopDownload

选项	说明	
描述	停止录像下载。	
函数	<pre>def StopDownload(cls, download_id: int ) -&gt; int</pre>	
参数	[in] download_id	下载接口 DownloadByTimeEx 返回值。
返回值	成功返回 1，失败返回 0。	
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。	



## 3.7 设备控制

### 3.7.1 获取配置信息函数 GetDevConfig

表3-21 获取配置信息函数 GetDevConfig

选项	说明	
描述	获取配置信息。	
函数	<pre>def GetDevConfig(cls, login_id: C_LLONG, cfg_type: C_DWORD, channel_id: C_LONG, out_buffer: C_LLONG, outbuffer_size: C_DWORD, wait_time: int = 5000 ) -&gt; int</pre>	
参数	[in] login_id	登录句柄,LoginWithHighLevelSecurity 返回值。
	[in] cfg_type	查询类型, 参考 SDK_Enum.py 文件内的 EM_DEV_CFG_TYPE 枚举。
	[in] channel_id	查询通道号。
	[out] out_buffer	获取的结构体数据。
	[in] outbuffer_size	out_buffer 数据长度。
	[in] wait_time	超时时间。
返回值	成功返回 1, 失败返回 0。	
说明	无。	

表3-22 配置类型枚举对照表

emType 枚举	含义
TIMECFG	时间配置, GetDevConfig 和 SetDevConfig 使用。

### 3.7.2 设置配置信息函数 SetDevConfig

表3-23 设置配置信息函数 SetDevConfig

选项	说明	
描述	设置配置信息。	
函数	<pre>def SetDevConfig(cls, login_id: C_LLONG, cfg_type: C_DWORD, channel_id: C_LONG, in_buffer: C_LLONG, inbuffer_size: C_DWORD, wait_time: int = 5000) -&gt; int</pre>	
参数	[in] login_id	登录句柄,LoginWithHighLevelSecurity 返回值。
	[in] cfg_type	查询类型, 参考 SDK_Enum.py 文件内的 EM_DEV_CFG_TYPE 枚举。
	[in] channel_id	查询通道号。

选项	说明	
	[in] in_buffer	传入的结构体数据。
	[in] inbuffer_size	in_buffer 数据长度。
	[in] wait_time	超时时间。
返回值	成功返回 1，失败返回 0。	
说明	无。	

### 3.7.3 远程设备重启函数 RebootDev

表3-24 远程设备重启函数 RebootDev

选项	说明	
描述	重启设备。	
函数	def RebootDev(cls, login_id: int ) -> int:	
参数	[in] login_id	LoginWithHighLevelSecurity 接口返回的登录句柄。
返回值	成功返回 1，失败返回 0。	
说明	无。	

## 3.8 远程抓图

### 3.8.1 设置远程抓图数据回调函数 SetSnapRevCallBack

表3-25 设置远程抓图数据回调函数 SetSnapRevCallBack

选项	说明	
描述	设置远程抓图数据回调函数。	
函数	def SetSnapRevCallBack(cls, OnSnapRevMessage: fSnapRev, dwUser: C_LDWORD ) -> None	
参数	[in] OnSnapRevMessage	前端远程抓图回调函数。
	[in] dwUser	用户数据，NetSDK 通过前端远程抓图回调函数 fSnapRev 将该数据返回给用户，以使用户后续操作。
返回值	无。	
说明	SetSnapRevCallBack 接口需在 SnapPictureEx 接口之前调用。	

### 3.8.2 抓图请求扩展接口 SnapPictureEx

表3-26 抓图请求扩展接口 SnapPictureEx

选项	说明
描述	抓图请求扩展接口。
函数	def SnapPictureEx(cls,

选项	说明	
	ILoginID:C_LLONG, par:SNAP_PARAMS, reserved=0 )->c_int	
参数	[in] ILoginID	LoginWithHighLevelSecurity 接口返回的登录句柄。
	[in] par	抓图参数，具体参见 SNAP_PARAMS 结构体说明。
	[in] reserved	保留字段。
返回值	成功返回 1，失败返回 0。	
说明	无。	

## 3.9 报警上报

### 3.9.1 设置报警回调函数接口 SetDVRMessCallBackEx1

表3-27 设置报警回调函数接口 SetDVRMessCallBackEx1

选项	说明	
描述	设置报警回调函数接口。	
函数	def SetDVRMessCallBackEx1(cls, cbMessage:fMessCallBackEx1, dwUser:C_LDWORD )->None	
参数	[in] cbMessage	报警回调函数，具体见 fMessCallBackEx1。
	[in] dwUser	用户数据，NetSDK 通过报警回调函数 fMessCallBackEx1 将该数据返回给用户，以便用户后续操作。
返回值	无。	
说明	SetDVRMessCallBackEx1 接口需在 StartListenEx 接口之前调用。	

### 3.9.2 订阅报警扩展接口 StartListenEx

表3-28 订阅报警扩展接口 StartListenEx

选项	说明	
描述	向设备订阅报警--扩展。	
函数	def StartListenEx(cls, ILoginID:C_LLONG )->c_int	
参数	[in] ILoginID	LoginWithHighLevelSecurity 接口返回的登录句柄
返回值	成功返回 1，失败返回 0。	
说明	所有设备的报警事件都是通过 SetDVRMessCallBackEx1 接口设置的回调函数反馈给用户的。	

### 3.9.3 停止订阅报警接口 StopListen

表3-29 停止订阅报警接口 StopListen

选项	说明	
描述	停止订阅报警接口。	
函数	def StopListen(cls, ILoginID:C_LLONG )->c_int	
参数	[in] ILoginID	LoginWithHighLevelSecurity 接口返回的登录句柄
返回值	成功返回 1，失败返回 0。	
说明	无。	

## 3.10 智能事件上报

### 3.10.1 智能图片报警订阅接口 RealLoadPictureEx

表3-30 智能图片报警订阅接口 RealLoadPictureEx

选项	说明	
描述	智能图片报警订阅接口。	
函数	def RealLoadPictureEx(cls, ILoginID: C_LLONG, nChannelID: c_int, dwAlarmType: c_ulong, bNeedPicFile: c_int, cbAnalyzerData: fAnalyzerDataCallBack, dwUser: C_LDWORD = 0, reserved: c_void_p = None ) -> C_LLONG	
参数	[in] ILoginID	LoginWithHighLevelSecurity 接口返回的登录句柄。
	[in] nChannelID	智能图片报警订阅通道号，通道号从 0 开始。
	[in] dwAlarmType	期望订阅的报警类型，参考 EM_EVENT_IVS_TYPE。
	[in] bNeedPicFile	是否订阅图片文件。 <ul style="list-style-type: none"><li>1：订阅图片</li><li>0：不订阅图片。</li></ul>
	[in]cbAnalyzerData	智能图片报警回调函数，当设备端有智能图片报警上报时，NetSDK 会调用该函数回调数据给用户 dwUser。
	[in] dwUser	用户数据，NetSDK 通过智能图片报警回调函数 fAnalyzerDataCallBack 将该数据返回给用户，以便用户后续操作。
	[in] reserved	保留参数。
返回值	成功返回智能图片报警订阅 ID，失败返回 0，将作为 StopLoadPic 的参数。	
说明	如果需要订阅一个通道上多种事件，那么请调用 RealLoadPictureEx 时，事件类型传 EM_EVENT_IVS_ALL，订阅所有事件类型，并在事件回调函数中	

选项	说明
	只处理想要处理的几种业务。

### 3.10.2 停止智能事件订阅接口 StopLoadPic

表3-31 停止智能事件订阅接口 StopLoadPic

选项	说明	
描述	取消智能图片报警订阅接口。	
函数	<pre>def StopLoadPic(cls, IAalyzerHandle:C_LLONG )-&gt;c_int</pre>	
参数	[in] IAnalyzerHandle	RealLoadPictureEx 接口返回的订阅句柄。
返回值	成功返回 1，失败返回 0。	
说明	无。	

## 第 4 章 回调函数定义

### 4.1 断线回调函数 fDisconnect

表4-1 fDisconnect

选项	说明
接口描述	断线回调函数。
前置条件	无。
函数	fDisconnect = WINFUNCTYPE(None, C_LLONG, c_char_p, c_long, C_LDWORD)
参数	<ul style="list-style-type: none"><li>• lLoginID: 登录句柄。</li><li>• pchDVRIP: IP 地址。</li><li>• nDVRPort: 端口号。</li><li>• dwUser: 用户数据。</li></ul>
返回值	无。
注释	此回调函数在设备断线情况下会被触发。 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。

### 4.2 断线重连回调函数 fHaveReConnect

表4-2 fHaveReConnect

选项	说明
接口描述	断线重连回调函数。
前置条件	无。
函数	fHaveReConnect = WINFUNCTYPE(None, C_LLONG, c_char_p, c_long, C_LDWORD)
参数	<ul style="list-style-type: none"><li>• lLoginID: 登录句柄。</li><li>• pchDVRIP: IP 地址。</li><li>• nDVRPort: 端口号。</li><li>• dwUser: 用户数据。</li></ul>
返回值	无。
注释	此回调函数会在断线的设备重新上线的情况下被触发。 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。

## 4.3 异步搜索设备回调函数 fSearchDevicesCBEx

表4-3 fSearchDevicesCBEx

选项	说明
接口描述	搜索设备回调原型。
前置条件	无。
函数	fSearchDevicesCBEx = WINFUNCTYPE(None, C_LLONG, POINTER(DEVICE_NET_INFO_EX2), c_void_p)
参数	<ul style="list-style-type: none"><li>• ISearchHandle: 搜索句柄。</li><li>• pDevNetInfo: 设备信息。</li><li>• pUserData: 用户数据信息。</li></ul>
返回值	无。
注释	搜索设备回调。 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。 通过 StartSearchDevicesEx 设置该回调函数，当搜索到设备时，NetSDK 会调用该函数。

## 4.4 搜索设备回调函数 fSearchDevicesCB

表4-4 fSearchDevicesCB

选项	说明
接口描述	搜索设备回调原型。
前置条件	无。
函数	fSearchDevicesCB = WINFUNCTYPE(None, POINTER(DEVICE_NET_INFO_EX), c_void_p)
参数	<ul style="list-style-type: none"><li>• pDevNetInfo: 信息。</li><li>• pUserData: 用户数据信息。</li></ul>
返回值	无
注释	搜索设备回调。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。 通过 SearchDevicesByIPs 设置该回调函数，当搜索到设备时，NetSDK 会调用该函数。

## 4.5 回放进度回调函数 fDownloadPosCallBack

表4-5 fDownloadPosCallBack

选项	说明
接口描述	回放进度回调函数。
前置条件	无。
函数	fDownloadPosCallBack = WINFUNCTYPE(None, C_LLONG, C_DWORD, C_DWORD, C_LDWORD)

选项	说明
参数	<ul style="list-style-type: none"> <li>● IPlayHandle: PlayBackByTimeEx 接口返回的句柄。</li> <li>● dwTotalSize: 下载数据总大小。</li> <li>● dwDownloadSize: 当前已下载数据的大小。 <ul style="list-style-type: none"> <li>◇ dwDownloadSize == -1, 表示用户回放或者下载进度完成。</li> <li>◇ dwDownloadSize == -2, 表示用户没有回放或者下载操作权限。</li> </ul> </li> <li>● dwUser: 用户数据。</li> </ul>
返回值	无。
注释	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。

## 4.6 回放数据回调 fDataCallBack

表4-6 fDataCallBack

选项	说明
接口描述	回放数据回调函数。
前置条件	无。
函数	fDataCallBack = WINFUNCTYPE(c_int, C_LLONG, C_DWORD, POINTER(c_ubyte), C_DWORD, C_LDWORD)
参数	<ul style="list-style-type: none"> <li>● IRealHandle: 回放数据句柄。</li> <li>● dwDataType: 数据类型。</li> <li>● pBuffer: 数据缓冲区, 内存由 NetSDK 内部申请释放。</li> <li>● dwBufSize: 数据缓存大小。</li> <li>● dwUser: 用户数据。</li> </ul>
返回值	<ul style="list-style-type: none"> <li>● 返回 1, 表示正常回调。</li> <li>● 返回 0: 表示接口数据阻塞, 下一次还是返回同样的码流数据。</li> </ul>
注释	下载录像时, 下载录像的数据回调函数。 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。

## 4.7 按时间下载进度回调函数 fTimeDownloadPosCallBack

表4-7 fTimeDownloadPosCallBack

选项	说明
接口描述	按时间下载进度回调函数。
前置条件	无。
函数	fTimeDownloadPosCallBack = WINFUNCTYPE(None, C_LLONG, C_DWORD, C_DWORD, c_int, NET_RECORDFILE_INFO, C_LDWORD)
参数	<ul style="list-style-type: none"> <li>● IPlayHandle: DownloadByTimeEx 接口返回的句柄。</li> <li>● dwTotalSize: 下载数据总大小。</li> <li>● dwDownloadSize: 当前已下载数据的大小。</li> </ul>



选项	说明
	<ul style="list-style-type: none"> <li>● Index: 文件序列。</li> <li>● Recordfileinfo: 录像文件信息。</li> <li>● dwUser: 用户数据。</li> </ul>
返回值	无。
注释	录像下载时，下载进度回调函数。 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。

## 4.8 智能图片报警回调函数 fAnalyzerDataCallBack

表4-8 fAnalyzerDataCallBack

选项	说明
接口描述	智能图片报警回调原型。
前置条件	无。
函数	fAnalyzerDataCallBack = WINFUNCTYPE(None, C_LLONG, C_DWORD, c_void_p, POINTER(c_ubyte), C_DWORD, C_LDWORD, c_int, c_void_p)
参数	<ul style="list-style-type: none"> <li>● lAnalyzerHandle: RealLoadPictureEx 接口返回的句柄。</li> <li>● dwAlarmType: EM_EVENT_IVS_TYPE 事件类型。</li> <li>● pAlarmInfo: 事件信息。</li> <li>● pBuffer: 图片数据缓存。</li> <li>● dwBufSize: 图片数据缓存大小。</li> <li>● dwUser: RealLoadPictureEx 输入的用户数据信息。</li> <li>● nSequence: 表示上传的相同图片情况, <ul style="list-style-type: none"> <li>◇ 0: 表示是第一次出现。</li> <li>◇ 1: 表示此次之后还有。</li> <li>◇ 2: 表示最后一次出现或仅出现一次。</li> </ul> </li> <li>● reserved: int nState = (int)reserved 表示当前回调数据的状态。 <ul style="list-style-type: none"> <li>◇ 0: 表示当前数据为实时数据。</li> <li>◇ 1: 表示当前回调数据是离线数据</li> <li>◇ 2: 表示离线数据传送结束。</li> </ul> </li> </ul>
返回值	无。
注释	<p>智能图片报警回调。</p> <p>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。</p> <p>通过 RealLoadPictureEx 设置该回调函数，当设备端有智能图片事件上报时，NetSDK 会调用该函数。</p> <p>dwAlarmType 值不同，pAlarmInfo 指向的数据类型不同。</p>

## 4.9 抓图回调函数 fSnapRev

表4-9 fSnapRev

选项	说明
接口描述	抓图回调函数原型
前置条件	无
函数	fSnapRev = WINFUNCTYPE(None, C_LLONG, POINTER(c_ubyte), c_uint, c_uint, C_DWORD, C_LDWORD)
参数	<ul style="list-style-type: none"><li>• ILoginID: 登录句柄。</li><li>• pBuf: 图片缓存。</li><li>• RevLen: 图片大小。</li><li>• EncodeType: 编码类型, 10: 表示 jpeg 图片 0: mpeg4 的 i 帧。</li><li>• CmdSerial: 请求填的序号。</li><li>• dwUser: SetSnapRevCallBack 接口输入的用户数据信息。</li></ul>
返回值	无
注释	抓图回调函数。 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。 通过 SetSnapRevCallBack 设置该回调函数, 当前端设备有抓图数据发送过来时, NetSDK 会调用该函数。

## 4.10 报警上报回调函数 fMessCallBackEx1

表4-10 fMessCallBackEx1

选项	说明
接口描述	报警上报回调函数原型。
前置条件	无。
函数	fMessCallBackEx1 = WINFUNCTYPE(None, c_long, C_LLONG, POINTER(c_char), C_DWORD, POINTER(c_char), c_long, c_int, c_long, C_LDWORD)
参数	<ul style="list-style-type: none"><li>• ICommand: 报警类型。</li><li>• ILoginID: 登录句柄。</li><li>• pBuf: 报警信息。</li><li>• dwBufLen: 报警信息大小。</li><li>• pchDVRIP: IP 地址。</li><li>• nDVRPort: 端口号。</li><li>• bAlarmAckFlag<ul style="list-style-type: none"><li>◇ 1: 表示该事件为可以进行确认的事件。</li><li>◇ 0: 该事件无法进行确认。</li></ul></li><li>• nEventID: 用于对 AlarmAck 接口的入参进行赋值, 当 bAlarmAckFlag 为 1 时, 该数据有效。</li><li>• dwUser: SetDVRMessCallBackEx1 接口输入的用户数据信息。</li></ul>
返回值	无。

选项	说明
注释	<p>所有登录的设备统一使用一个报警上报回调函数。</p> <p>用户通过参数 <b>ILoginID</b> 来定位到是哪次登录对应的报警上报。</p> <p><b>ICommand</b> 值不同，<b>pBuf</b> 指向的数据类型不同。</p> <p>在该回调函数中不建议调用任何 <b>NetSDK</b> 接口。但如果 <b>Demo</b> 内该回调函数调用了 <b>NetSDK</b> 接口，则您可以同样处理。</p>

# 附录1 法律声明

## 商标声明

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

## 责任声明

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

## 隐私保护提醒

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

## 关于本文档

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

## 附录2 网络安全建议

### 保障设备基本网络安全的必须措施：

#### 1. 使用复杂密码

请参考如下建议进行密码设置：

- 长度不小于 8 个字符。
- 至少包含两种字符类型，字符类型包括大小写字母、数字和符号。
- 不包含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。