# NetSDK_JAVA 开发

## FAQ

V2.0.0

# 前言

## 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 |
| --- | --- | --- |
| V2.0.0 | 补充问题说明 | 2025.02 |
| V1.0.0 | 首次发布。 | 2021.04 |

# 目录

# 第 1 章 结构体封装问题

## 1.1 Pointer 类型作为入参时提示空指针

### 现象描述

结构体.getPointer()将结构体转为 pointer 类型作为入参传入接口时报空指针异常。

### 可能原因

调用接口时，参数直接使用了结构体对象。

### 解决方法

- 方法 1：对结构体先执行 write()操作，再执行 getPointer()操作，即可正常将结构体中的数据写入到指针中，最后调用接口时就不会出现空指针异常问题。
- 方法 2：使用 SDK 提供的工具类 ToolKits.java 中的 SetStructDataToPointer(单个结构体)、SetStructArrToPointerData（结构体数组）。

## 1.2 Pointer 类型作为出参时报空指针异常

### 现象描述

Pointer 类型作为方法的参数，解析为对应结构体时，报空指针异常。

### 可能原因

调用接口成功后，解析结构体时，直接使用了结构体对象。

### 解决方法

- 方法 1：先将 pointer 内存中的数据通过 read()方法读取到对应的结构体，再解析需要的字段。
- 方法 2：使用 SDK 提供的工具类 ToolKits.java 中的 GetStructDataToPointer(单个结构体)、GetStructArrToPointerData（结构体数组）。

## 1.3 JAVA 封装的结构体内存未对齐字段乱码

### 现象描述

修改 SDK 封装层或者替换新的动态库过程中，解析 JAVA 封装的结构体对象的字段时出现部分

乱码甚至完全乱码。

## 可能原因

读取内存数据时，读取了无效的内存。

## 解决方法

排查封装问题，确保封装的类与 C 结构体字节对齐且字段顺序一致。

# 1.4 程序报结构体字段不匹配

## 现象描述

报错如下：

```
java.lang.reflect.InvocationTargetException
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:497)
    at com.netsdk.demo.util.CaseMenu.run(CaseMenu.java:78)
    at
com.netsdk.demo.customize.querySystemState.QuerySystemStateDemo.runTest(QuerySyste
mStateDemo.java:230)
    at
com.netsdk.demo.customize.querySystemState.QuerySystemStateDemo.main(QuerySystem
StateDemo.java:251)
Caused by: java.lang.Error: Structure.getFieldOrder() on class
com.netsdk.lib.structure.NET_SYSTEM_STATUS does not provide enough names [0] ([]) to
match declared fields.
```

## 可能原因

java 类继承了 Structure。

## 解决方法

JAVA 封装的结构体类继承了错误的父类（Structure），继承 SdkStructure 即可。

# 1.5 Pointer 与 JAVA 对象怎么相互转换？

## 现象描述

遇到 Pointer 参数或者变量时，不清楚如何和 JAVA 封装的结构体转换；或者是不清楚如何将 JAVA 封装的结构体对象赋值给 Pointer。

## 解决方案

使用如下任意一种方法解决。

- 方法 1：对象转 Pointer 时，申请相同大小的内存块，使用工具类 ToolKits.java 中的 Set 开头的方法赋值。

eg:

```
CFG_SPLIT_INFO pSplits = new CFG_SPLIT_INFO();
pSplits.pSplitChannels = new Memory(pSplitChannels.size());
pSplits.emSplitMode = 1;
pSplits.nReturnChannels = 1;
http://wereview.dahuatech.com/Review/ReviewApproval?ReviewID=90801
pSplits.nMaxChannels = 1;
ToolKits.SetStructDataToPointer(pSplitChannels, pSplits.pSplitChannels, 0);
```

- 方法 2：读取 Pointer 中的数据时，先 new JAVA 对象，使用工具类 ToolKits.java 中的 Get 开头的方法获取。

eg:

```
ALARM_VEHICLE_INOUT_INFO stuAlarmInfo = new ALARM_VEHICLE_INOUT_INFO();
ToolKits.GetPointerDataToStruct(pStuEvent, 0, stuAlarmInfo);
```

# 第 2 章 动态库加载问题

## 2.1 如何加载 C 语言动态库？

### 现象描述

Windows 平台动态库加载失败。

```
java.lang.UnsatisfiedLinkError: Unable to load library '***': Native library (win32-x86-64/jndadll.dll) not found in resource path
```

### 可能原因：

动态库未加载或者加载失败。

### 解决方法

- 方法 1：将动态库所在文件夹按“操作系统-操作系统位数”重命名，如 win32-x86,win32-x86-64,linux-x86,linux-x86-64 等，并修改 NetSDKLib.java 中的加载路径为：NetSDKLib NetSDK_INSTANCE = Native.load("dhnetsdk", NetSDKLib.class);NetSDKLib CONFIG_INSTANCE = Native.load("dhconfigsdk", NetSDKLib.class);
  该种方式原理为从工程中读取动态库，写入 java.io.tmpdir 对应的路径（linux 下一般为/tmp），从该路径中加载库。需要保证动态库所在文件夹不能重名，即 windows64 下为 win64，linux64 下为 linux64。
- 方法 2：使用 LibraryLoad.java 加载，该种方式原理为从工程中读取动态库，并写入 java.io.tmpdir 对应的路径（linux 下一般为/tmp），从该路径中加载库。需要保证动态库所在文件夹不能重命名，即 windows 操作系统 64 位下为 win64，linux 系统 64 位下为 linux64。

## 2.2 Linux 操作系统如何加载动态库？

### 现象描述

Linux 平台使用 NetSDK，启动项目报无法加载动态库错误。

### 可能原因

动态库加载失败。

### 解决方法

使用环境变量 LD_LIBRARY_PATH，使用脚本方式加载：

Export LD_LIBRARY_PATH=/path/to/so

java –jar myproject.jar

启动工程前，将动态库所在路径添加到环境变量 LD_LIBRARY_PATH 中。

# 2.3 Linux 下报错 Invalid calling convention 63

## 现象描述

完整报错如下：

```
java.lang.IllegalArgumentException: Invalid calling convention 63
        at com.sun.jna.Native.createNativeCallback(Native Method)
        at com.sun.jna.CallbackReference.<init>(CallbackReference.java:263)
        at com.sun.jna.CallbackReference.getFunctionPointer(CallbackReference.java:449)
        at com.sun.jna.CallbackReference.getFunctionPointer(CallbackReference.java:426)
        at com.sun.jna.Function.convertArgument(Function.java:558)
        at com.sun.jna.Function.invoke(Function.java:345)
        at com.sun.jna.Library$Handler.invoke(Library.java:265)
        at com.sun.proxy.$Proxy10.CLIENT_Init(Unknown Source)
        at com.netsdk.lib.LoginModule.init(LoginModule.java:34)
        at
com.yfqc.facevideo.common.service.CameraInfoService.<init>(CameraInfoService.java:59)
    at
com.yfqc.facevideo.common.task.CheckCameraInfoTask.doUpdate(CheckCameraInfoTask.java:46)
        at
com.yfqc.facevideo.common.task.CheckCameraInfoTask.run(CheckCameraInfoTask.java:30)
        at it.sauronsoftware.cron4j.RunnableTask.execute(Unknown Source)
        at it.sauronsoftware.cron4j.TaskExecutor$Runner.run(Unknown Source)
at java.lang.Thread.run(Thread.java:745)
```

## 可能原因

Linux 平台使用了 Windows 版的 NetSDKLib.java。

## 解决方法

使用 Linux 平台的 NetSDKLib.java 文件。

# 2.4 Linux 下加载 SO 文件报 undefined symbol: SP_LoadLibrary

## 现象描述

Linux 下运行程序时报错：undefined symbol: SP_LoadLibrary。

## 可能原因

动态库缺少相关函数或者宏的依赖。

## 解决方法

在动态库所在路径，Linux 终端上输入 ldd -r *.so，查看依赖的函数或者宏，更换并使用新的动态库。

# 第 3 章 其他问题

## 3.1 使用回调函数导致程序崩溃

### 现象描述

JAVA 开发人员通常很少接触、使用回调，对于 NetSDK 中的回调不太熟悉，用回调导致程序崩溃。

### 可能原因

回调被垃圾回收、网络断线。

### 解决方法

回调是指通过函数参数传递到其它代码的，某一块可执行代码的引用。这一设计允许了底层代码调用在高层定义的子程序，参考 NetSDK 中的回调使用。

也可以这样理解回调：可把回调函数理解为一个 websocket 长连接，比如说 A 类中的方法里调用了 B 类中的方法，相当与 A、B 服务器间建立了长连接，A 实时监听 B 响应后发来的消息，并进行处理，即 A 中的回调函数的触发是由 B 来决定的。回调函数典型的用法就是支付宝支付，发来消息，银行转账，而后银行又发来消息，这里支付宝调用了银行的回调函数。

回调 Pointer 赋值，用自定义的数组暂存，然后从数组中取数据替代直接去取 Pointer 中的值，以免 Pointer 被回收再读取数据崩溃。

**注意：**

- 回调函数中不能调用 SDK 接口，需要另起一个线程执行。
- 回调方法是弱引用，而回调是异步的，所以应该写成单例或全局形式，防止被 JVM 回收。
- 回调是异步的，底层 C++基本都是会加锁的。为了防止死锁，回调里不应该调用 NetSDK 其他方法。

## 3.2 错误码封装

### 现象描述

调用 C 层接口时，返回的错误码表意不明。

### 解决方法

错误码封装位置 com.netsdk.lib.enumeration.ENUMERROR，可使用 getErrorMessage 打印错误码所对应完整描述信息。

eg:

```
System.out.println("query system info failed.error is" + ENUMERROR.getErrorMessage());
```

## 3.3 动态库和封装层的库版本不匹配，乱码、崩溃

### 现象描述

解析数据时字段值不对、部分乱码、全乱码、JNA 层崩溃。

### 可能原因

客户项目中替换了其他版本的动态库，JAVA 封装的结构体与动态库中的结构体没有匹配。

### 解决方法

按照对应的库和头文件封装 JAVA 类（确保 JAVA 封装类、dll 或者 so 与头文件一致）。

## 3.4 Linux 平台 NetSDK Jar 包依赖报缺少动态库依赖

### 问题描述

转码需求 Linux 平台下载录像文件出错，接口调用失败。

### 可能原因

缺少转码库依赖。

### 解决方法

通常 Windows 平台下无此问题，添加转码库： 使用脚本启动，参考 NetSDK 中的 run.sh 文件。

## 3.5 不同的操作系统平台编码格式如何设置？

### 现象描述

解析数据时，数据乱码。

### 可能原因

编码格式设置错误。

### 解决方案

Windows 平台设置编码格式为 GBK，Linux 平台设置编码格式为 UTF-8。

## 3.6 回调中获取 dwUser 数据时乱码或者报 Invalid Memory Access 错误

### 现象描述

回调中获取 dwUser 乱码或者出错。

### 可能原因

dwUser 被垃圾回收。

### 解决方法

自定义数据传入 null。使用 dwUser 时，将其声明为全局变量，保证 dwUser 不被 jvm 回收。

## 3.7 什么是句柄？

JAVA 开发人员很少接触、使用句柄。句柄是一个唯一的整数，作为对象的身份 id，用于区分不同的对象和同类中的不同实例。程序可以通过句柄访问对象的部分信息。

JAVA 中任何东西都可以视为对象。我们可以认为操纵的标识符实际是指向一个对象的“句柄”(Handle)。

## 3.8 字符串传参、提取导致崩溃

### 现象描述

下发字符串数据，导致设备崩溃。

### 可能原因

编码格式设置错误。

### 解决方法

- Linux 平台字符串编码格式为 UTF-8，传参、提取时需指定格式。
- Windows 平台编码格式为 GBK，传参、提取时需指定。

# 3.9 日志怎么打开，路径怎么指定？

## 现象描述

在使用 JAVA 封装的 SDK 遇到问题，需借助日志分析、定位问题时，不清楚有日志模块或者不清楚怎么开启。

## 解决方案

需在相关代码模块中封装 SDK 日志模块功能，路径可自行指定。如下是相关示例代码：

```
/**
 * 打开 sdk log
 */
public static void enableLog() {
    NetSDKLib.LOG_SET_PRINT_INFO setLog = new
NetSDKLib.LOG_SET_PRINT_INFO();
    File path = new File("sdklog/");
    if (!path.exists()) path.mkdir();

    // 这里的 log 保存地址依据实际情况自己调整
    String logPath = path.getAbsoluteFile().getParent() + "\\sdklog\\" + "sdklog" +
AnalyseTaskUtils.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");
}
```

# 3.10 导入项目在 idea 中 NetSDKLib 类报红

## 现象描述

初次在 idea 中导入项目后发现 NetSDKLib 类未成功引入项目，导致程序无法正常运行

## 解决方案

打开上方的 Help->Edit Custom Properties…，复制 idea.max.intellisense.filesize=4096 这段代码进入文件，关闭所有的 idea 编译器，再重新打开即可完成配置

# 3.11 订阅事件无法持续接收

## 现象描述

订阅事件初期可接收，但一段时间后失效

## 解决方案

回调事件需要使用静态单例模式，防止在程序运行时被系统回收导致回调失效。如下用智能订阅事件作为相关示例代码：

```
/**
 * 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;
    //EVENT_IVS_ALL 表示订阅所有智能事件
    m_attachHandle = netsdk.CLIENT_RealLoadPictureEx(loginHandle, channelId,
EVENT_IVS_ALL, bNeedPicture, AnalyzerDataCB.getInstance(), null, null);
    /**
     * // EVENT_IVS_WORKCLOTHES_DETECT 安全帽检测事件
     * // EVENT_IVS_SMOKING_DETECT 吸烟检测事件
     **/
    if (m_attachHandle.longValue() != 0) {
        System.out.printf("Chn[%d] CLIENT_RealLoadPictureEx Success\n", channelId);
    } else {
        System.out.printf("Ch[%d] CLIENT_RealLoadPictureEx Failed!LastError = %s\n",
channelId,
                ToolKits.getErrorCode());
    }
}


/**
 * 报警事件（智能）回调
 */
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;
        }

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

            switch (dwAlarmType) {
                case EVENT_IVS_WORKCLOTHES_DETECT: // 安全帽检测事件
                {
                    NetSDKLib.DEV_EVENT_WORKCLOTHES_DETECT_INFO msg = new
NetSDKLib.DEV_EVENT_WORKCLOTHES_DETECT_INFO();
                    ToolKits.GetPointerData(pAlarmInfo, msg);
                    if (msg.stuSceneImage != null && msg.stuSceneImage.nLength > 0) {
                        String bigPicture = picturePath + "\\" + System.currentTimeMillis() +
".jpg";
                        ToolKits.savePicture(pBuffer, msg.stuSceneImage.nOffSet,
msg.stuSceneImage.nLength, bigPicture);
                        if (msg.stuHumanImage != null && msg.stuHumanImage.nLength > 0) {
                            String smallPicture = picturePath + "\\" + System.currentTimeMillis()
+ "small.jpg";
                            ToolKits.savePicture(pBuffer, msg.stuHumanImage.nOffSet,
msg.stuHumanImage.nLength, smallPicture);
                        }
                    }
                    System.out.println(" 安全帽检测事件(UTC)：" + msg.UTC + " 通道号:" +
msg.nChannelID);
                    break;
                }
                case NetSDKLib.EVENT_IVS_SMOKING_DETECT : {      // 吸烟检测事件(对应
DEV_EVENT_SMOKING_DETECT_INFO)
                    System.out.printf("吸烟检测事件");
                    DEV_EVENT_SMOKING_DETECT_INFO msg = new
DEV_EVENT_SMOKING_DETECT_INFO();
                    ToolKits.GetPointerData(pAlarmInfo, msg);
                    String Picture = picturePath + "\\" + "smoking_" + System.currentTimeMillis()
+ ".jpg";

                    if (dwBufSize > 0) {
                        ToolKits.savePicture(pBuffer,   msg.stuImageInfo[0].nOffset,
msg.stuImageInfo[0].nLength, Picture);
                    }
```

```
                    System.out.println("吸烟检测事件 时间(UTC)：" + msg.UTC + " 开始时间:" +
msg.stuObject.stuStartTime + " 结束时间:"
                            + msg.stuObject.stuEndTime);
                    break;
                }
                case EVENT_IVS_GRANARY_TRANS_ACTION_DETECTION:{ //粮面异动检测
事件上报(对应 NET_DEV_EVENT_GRANARY_TRANS_ACTION_DETECTION_INFO)
                    System.out.println("粮面异动检测事件上报");
                    NET_DEV_EVENT_GRANARY_TRANS_ACTION_DETECTION_INFO msg
= new NET_DEV_EVENT_GRANARY_TRANS_ACTION_DETECTION_INFO();
                    ToolKits.GetPointerData(pAlarmInfo, msg);
                    System.out.println("粮面异动检测事件上报 时间(stuUTC)：" + msg.stuUTC);
                    System.out.println("物体列表消息实际数量（nObjectsCount）：" +
msg.nObjectsCount);
                    break;
                }
                case EVENT_IVS_REGION_PROPORTION_DETECTION:{ // 区域占比检测事件
(对应 NET_DEV_EVENT_REGION_PROPORTION_DETECTION_INFO)
                    System.out.println("区域占比检测事件");
                    NET_DEV_EVENT_REGION_PROPORTION_DETECTION_INFO msg =
new NET_DEV_EVENT_REGION_PROPORTION_DETECTION_INFO();
                    ToolKits.GetPointerData(pAlarmInfo, msg);
                    System.out.println("区域占比检测事件上报 时间(stuUTC)：" + msg.stuUTC);
                    System.out.println("物体列表消息实际数量（nObjectsCount）：" +
msg.nObjectsCount);
                    break;
                }
                default:
                    System.out.println("其他事件----------"+ dwAlarmType);
                    break;
            }
            return 0;
        }
}
```

# 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

- 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。