工业相机 SDK 开发手册

（C 篇）

Ver2.0

变更履历

编号	变更日期	版本号	变更内容概要	配套 SDK 版本
1	2022/02	V2.0	参照修改	2.3.1

目录

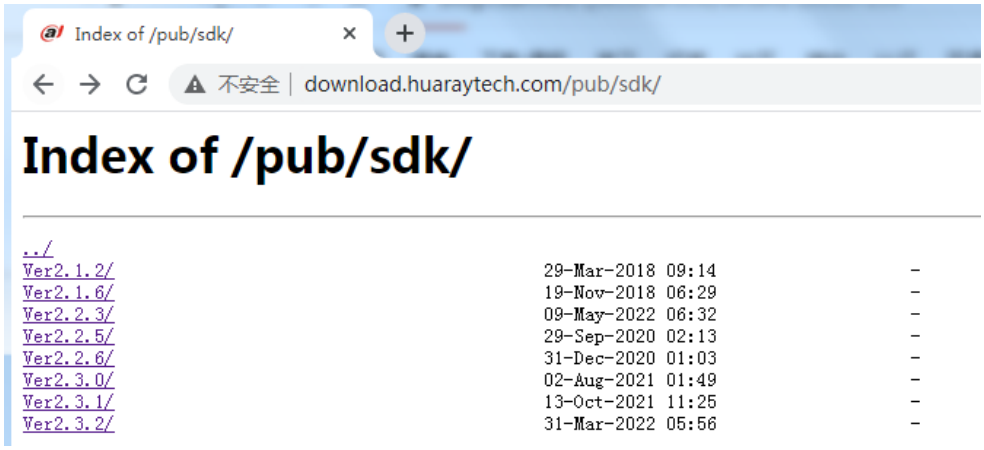
1. SDK 概要说明

1.1. ••下载 SDK 安装包

大华工业相机 SDK 可以通过华睿官网下载。

下载链接 ： http://download.huaraytech.com/pub/sdk/

在该网页内，罗列了各个版本的 SDK。根据需要，下载相应的版本。



1.2. SDK 支持的操作系统

SDK 支持以下版本的操作系统

操作系统类型	操作系统版本号	备注
Windows *1	Win7 SP1 32/64bit	务必注意带 SP1 补丁
	Win8 32/64bit	
	Win10 32/64bit	
Linux (x86) *2	系统版本： Ubuntu 12.04 LTS ~ Ubuntu 21.04 LTS （32/64bit）	不包括升级了内核版本的系统
	CentOS 6.5 ~ CentOS 8.0 （32/64bit）	不包括升级了内核版本的系统

*1. 从 SDK2.2.5 开始已经不支持 windows xp 系统

*2. 目前支持的 Linux 内核版本号从 2.6.32（含）到 5.11.0（含）。
可以通过 uname －a 命令来确定系统内核版本号。

1.3. SDK 安装目录

1.3.1. Windows 版 SDK

Windows 版本 SDK 的安装目录结构如下。

※ 从 Ver2.1.0 版本开始，用户可以自己选择安装位置。

文件夹说明如下：

Application	1.MV Viewer（相机客户端软件），包含了 32 位与 64 位 2.CamTools（相机配套工具软件），包含了 32 位与 64 位
Development	**3rdPartyPlatformAdapter** 包含了两个第三方算法平台（Halcon、Sherlock）插件动态库
	DOTENT DLL 包含.Net 库文件，在创建项目时从中导入
	Include 包含了 SDK 的接口头文件 ,头文件中解释了各个接口的基础属性，具体用法在 samples 文件夹中有示例

	另一个是 SDK 扩展定义头文件，定义了 SDK 的错误码、参数自定义类型、些许参数的数据限制、及各类行为方式的枚举、定义了多种信息的数据结构
	Lib 包含 lib 库文件
	Samples 包含 demo 示例（C#、Delphi、Lab View、MFC、Python、QT、VB、VC）
Documentations	.Net 接口文档 ActiveX 接口文档 C 接口文档 SDK 开发手册
Drivers	DShowFilter 的安装与卸载程序 GigE 高性能驱动的安装与卸载程序 PCIex86 智能相机驱动的安装与卸载程序 usb 高性能驱动的安装与卸载程序
Runtime	SDK 运行时编译动态链接库 包含第三方 GenICam 库

	分为 32 位与 64 位两个文件夹

1.3.2. Linux 版 SDK

Linux 版本 SDK 的安装目录固定在/opt/HuarayTech/MVviewer，文件夹结构、以及概要说明如下：

```
MVviewer
  ├ bin            : GUI工具，发现、连接相机，拉流取图，设置相机属性，卸载脚本，etc
  ├ include        : SDK头文件、转码头文件
  ├ lib            : SDK动态库、第三方库、转码库
  ├ module
  |  ├ GigEDriver : GigE相机驱动程序
  |  └ USBDriver  : USB相机的驱动程序
  └ share
     ├ C           : C例程
     ├ Qt          : Qt例程
     └ Python      : Python例程
```

1.4. SDK（c 接口）对应的资源

大类别	小类别	文件位置
SDK 库	头文件	【SDK 安装目录】\Development\Include\IMVApi.h 【SDK 安装目录】\Development\Include\IMVDefines.h
	Lib 库（32 位）	【SDK 安装目录】\Development\Lib\win32\MVSDKmd.lib
	Lib 库（64 位）	【SDK 安装目录】\Development\Lib\x64\MVSDKmd.lib
	运行库（32 位）	【SDK 安装目录】\Runtime \Win32\MVSDKmd.dll
	运行库（64 位）	【SDK 安装目录】\Runtime \x64\MVSDKmd.dll
C API 文档	中文	【SDK 安装目录】\Documentations\ GenICam_API_C_CHS.chm
	英文	【SDK 安装目录】\Documentations\ GenICam_API_C_ENG.chm

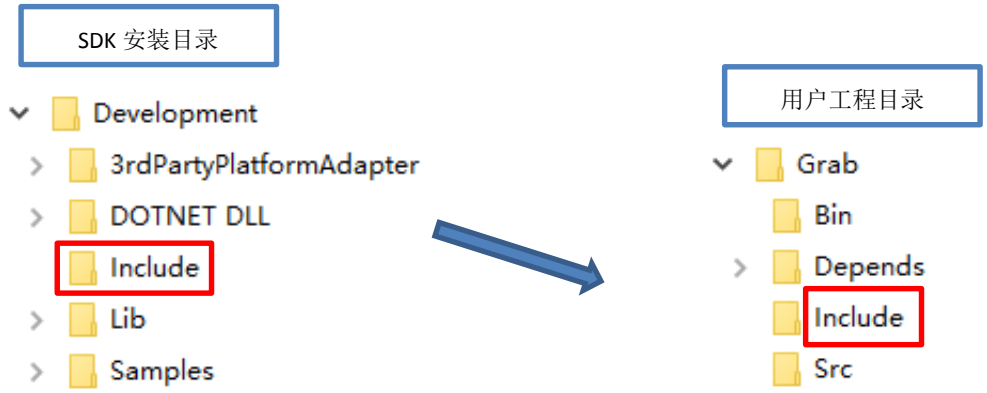
1.5. SDK 开发工程配置

1.5.1. VC6.0

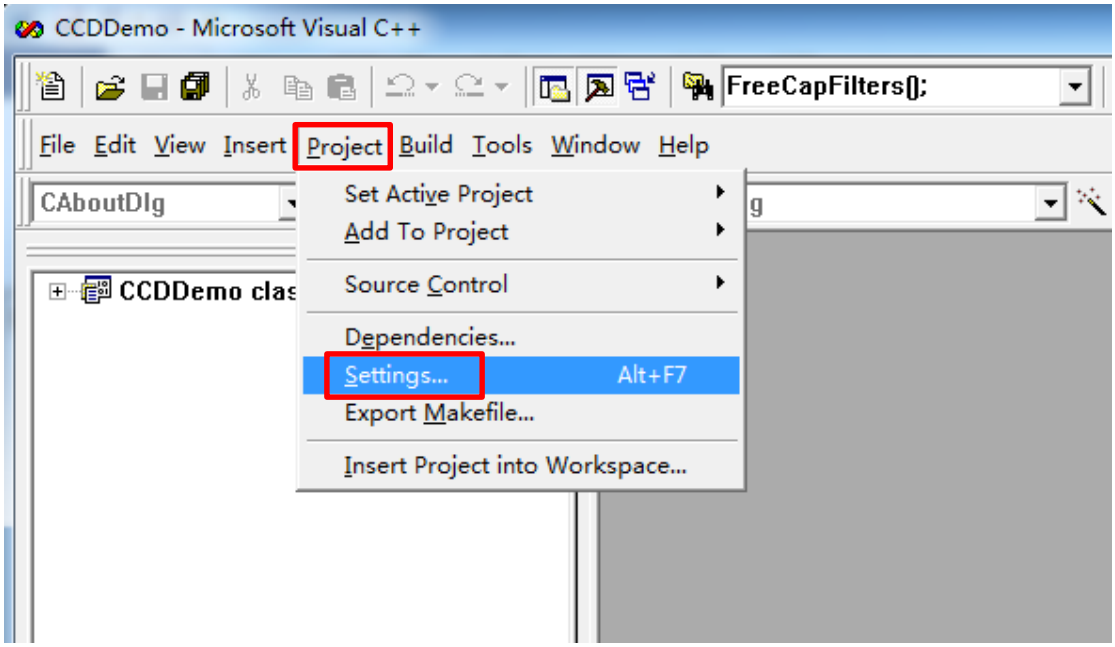
下面以 Win32|Debug 组合为例演示工程配置方法。

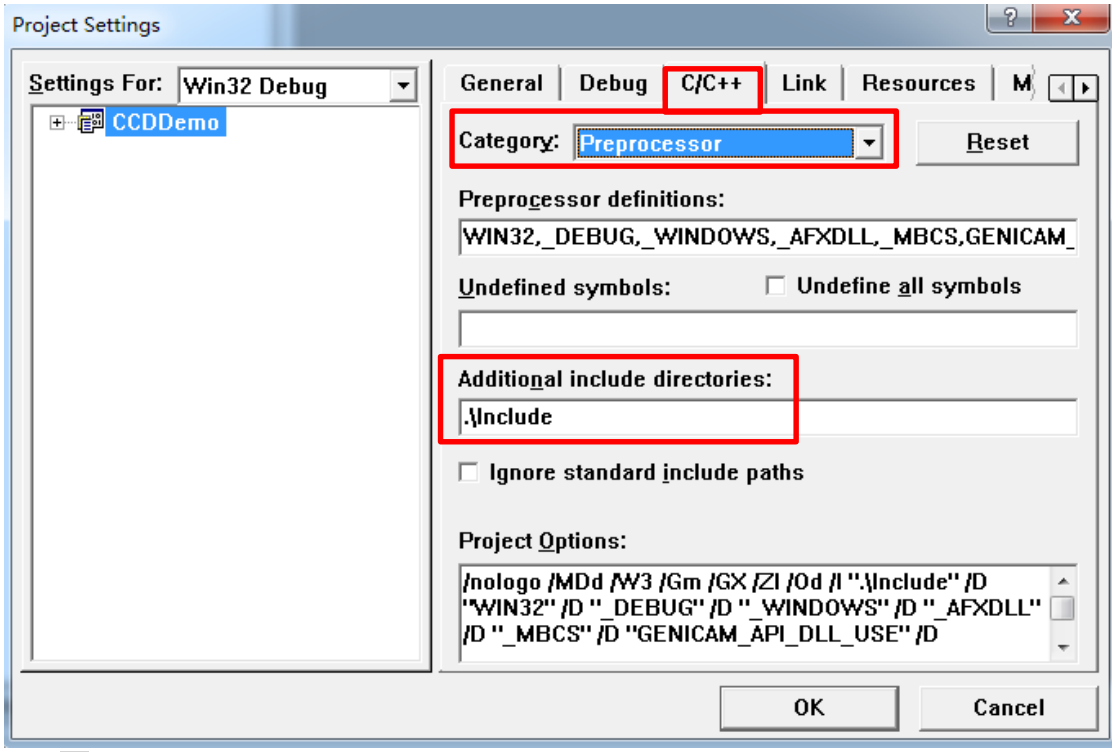
①配置头文件。

a.拷贝头文件



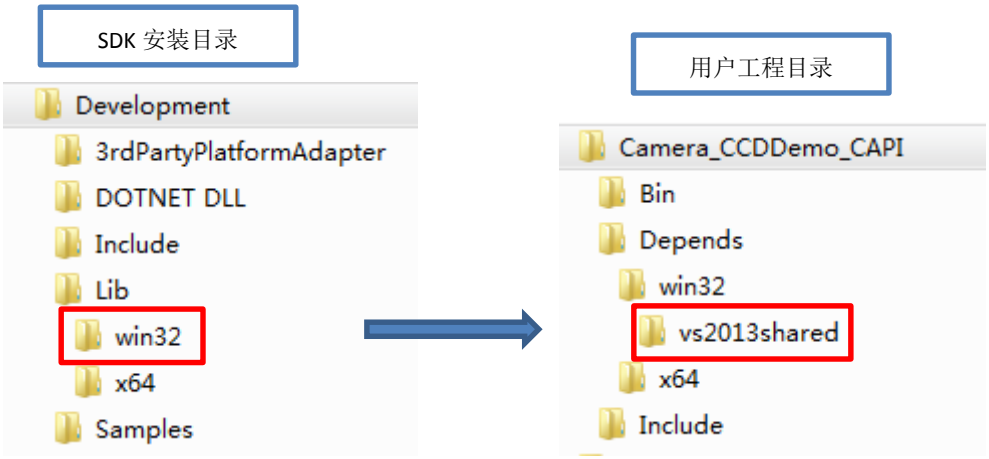
b.工程配置



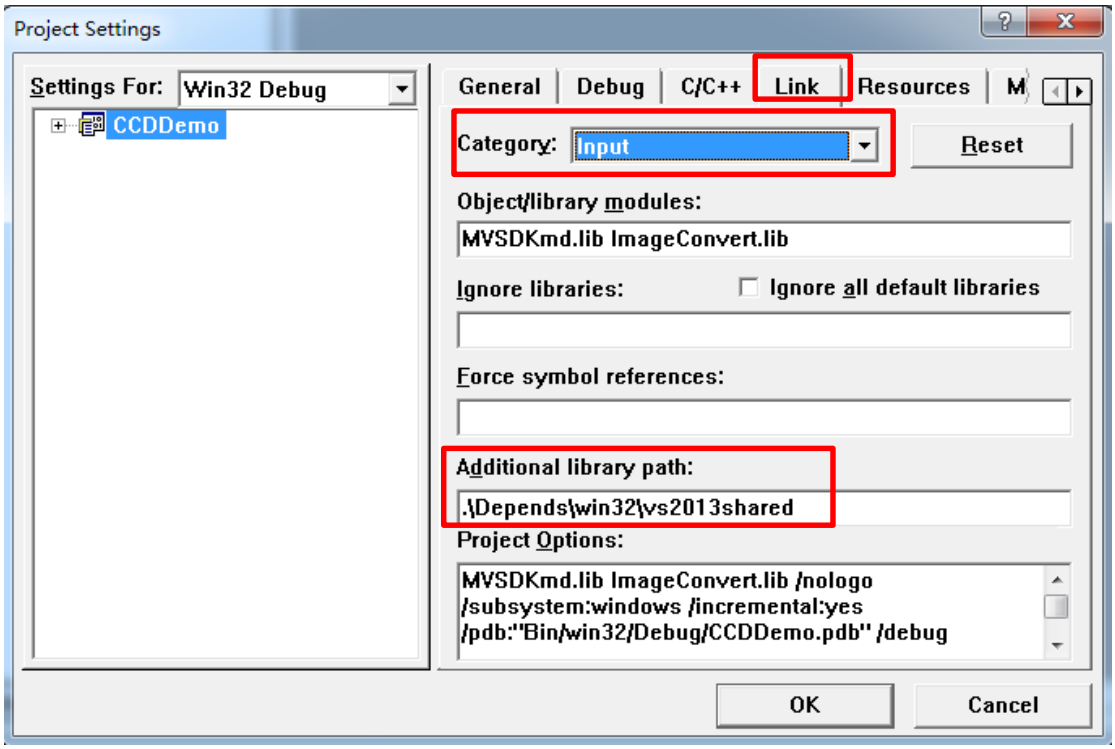


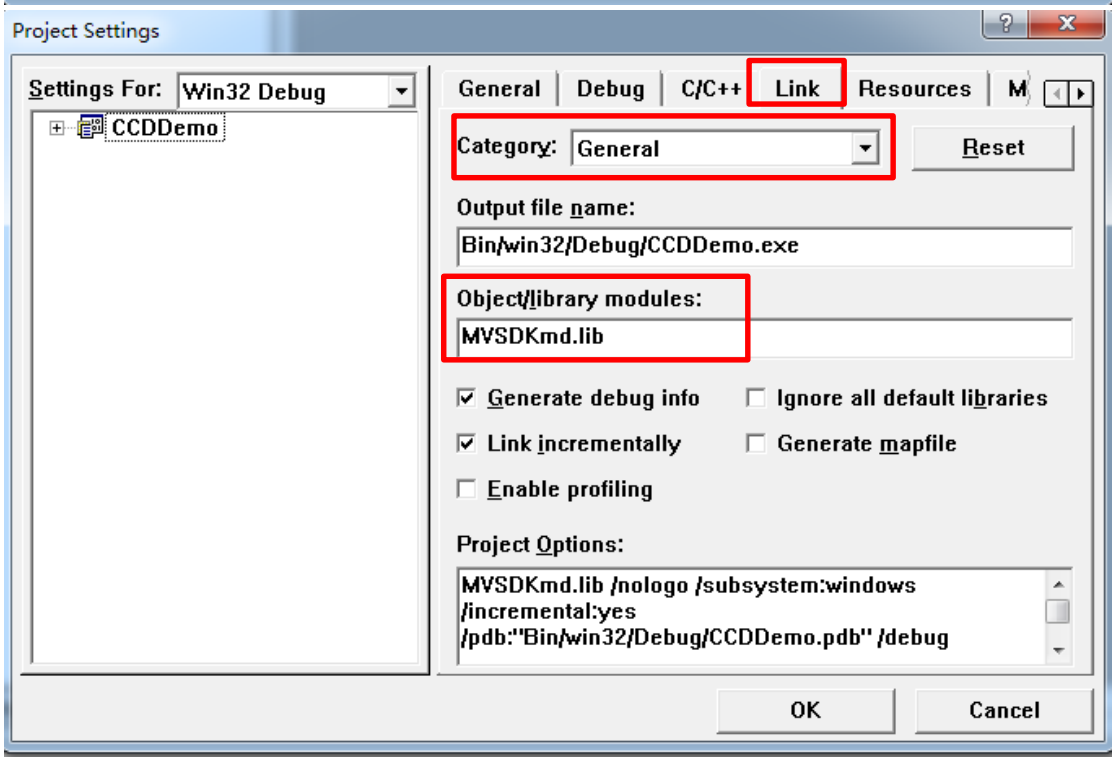
②配置 lib 库。

a.拷贝 MVSDKmd.lib



b.工程配置





c.运行库

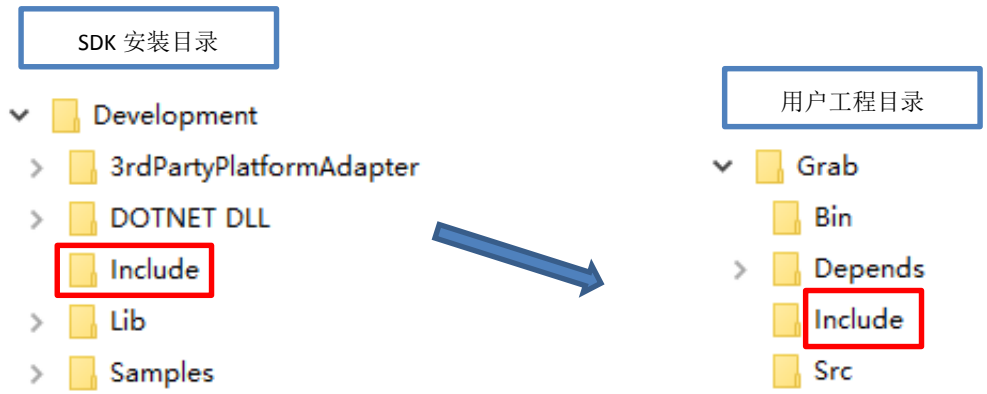
不需要拷贝 dll 库，默认会使用【SDK 安装目录】\Runtime 下对应平台的 dll。

1.5.2. VS2005 ~ VS2013

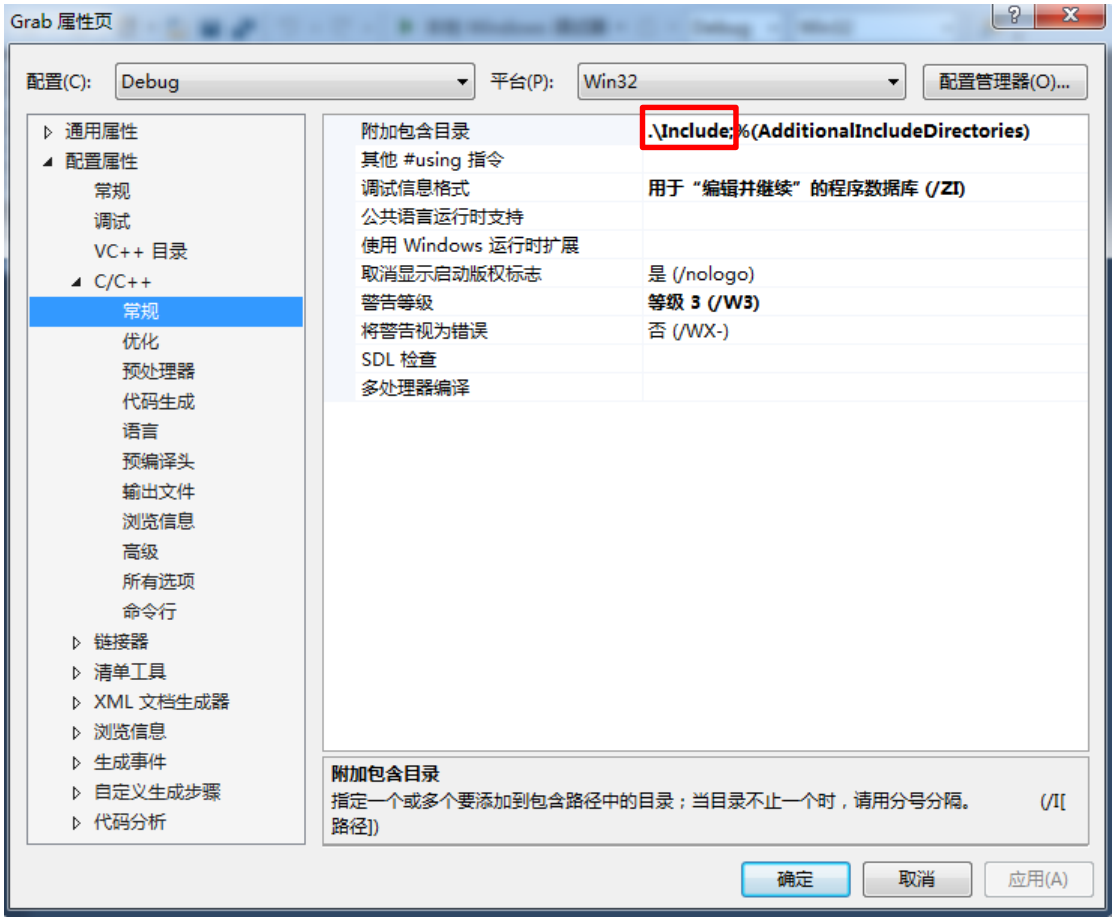
下面以 VS2013 Debug|Win32 组合为例演示工程配置方法。

①配置头文件。

a.拷贝头文件

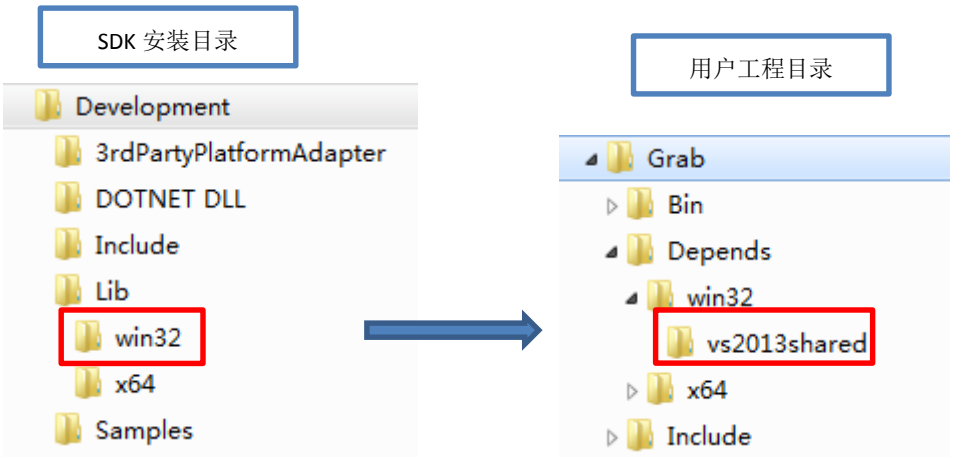


b.工程配置

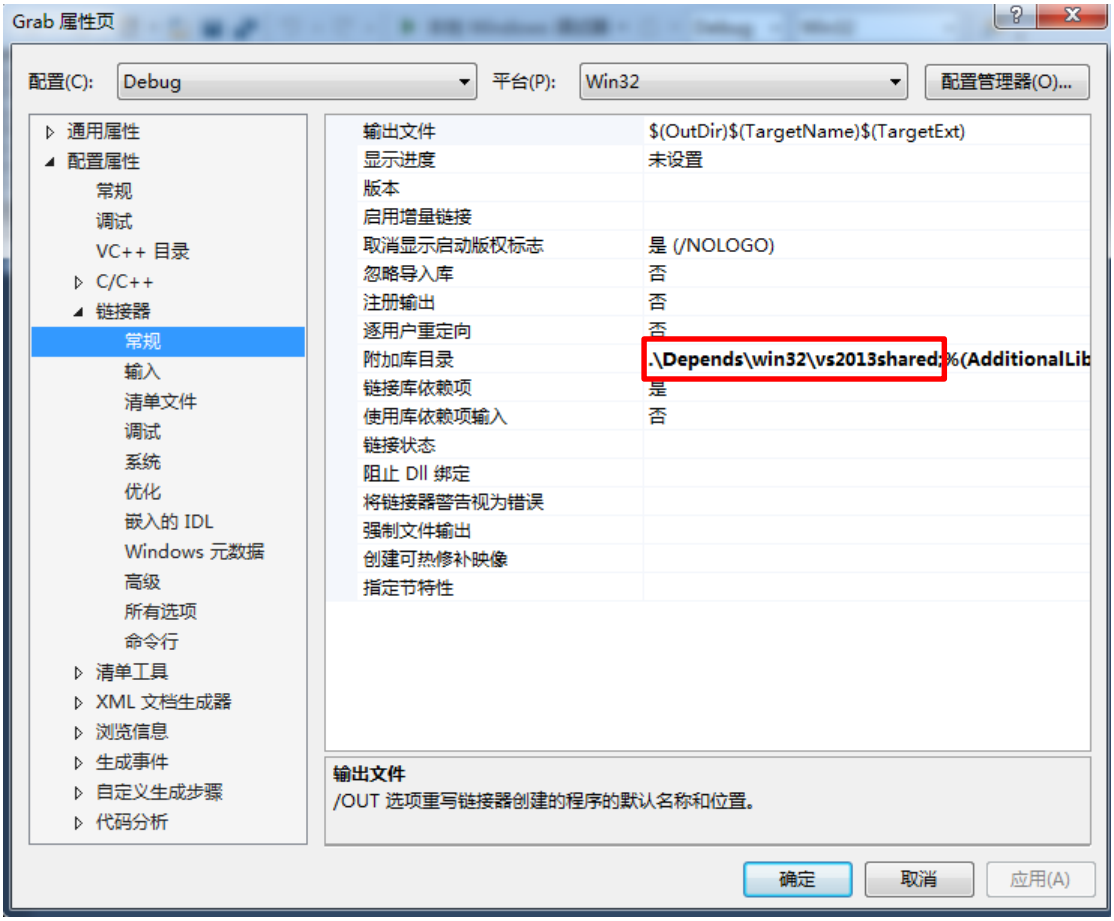


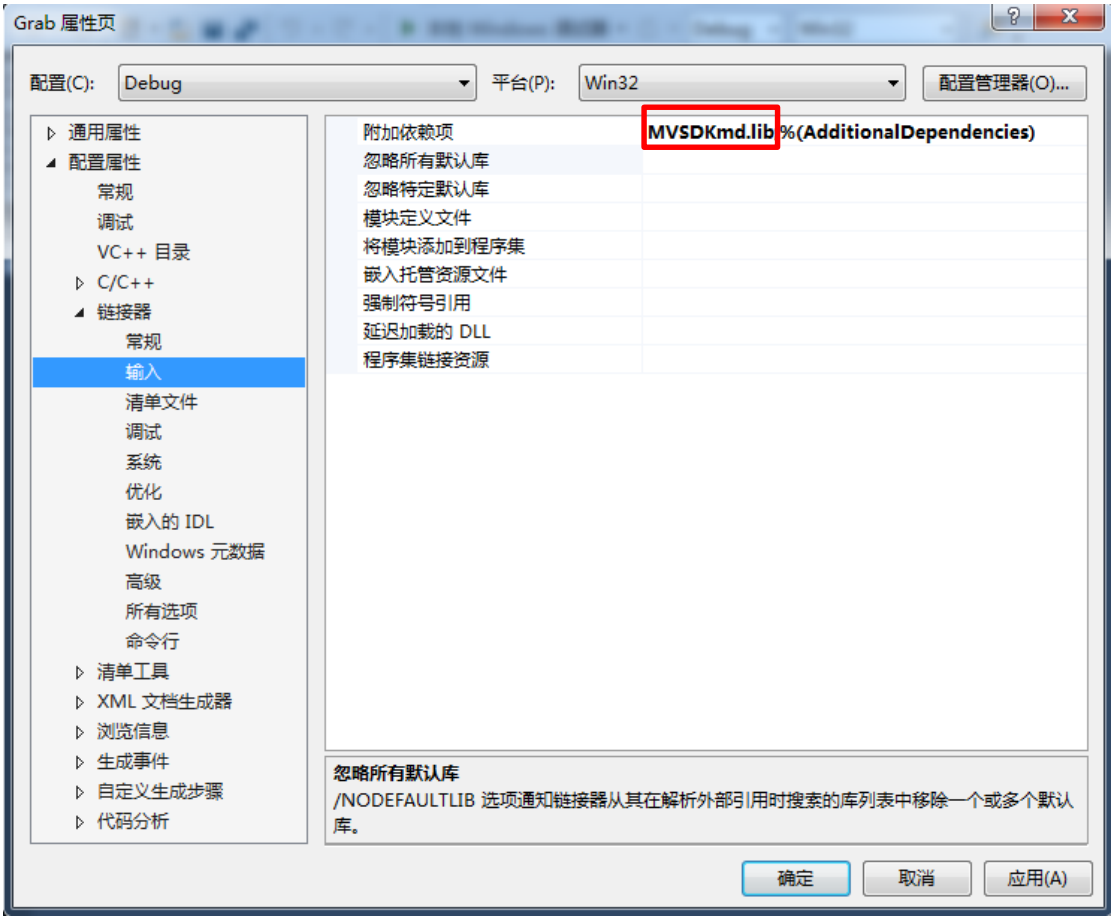
②配置 lib 库。

a.拷贝 lib 库



b.工程配置



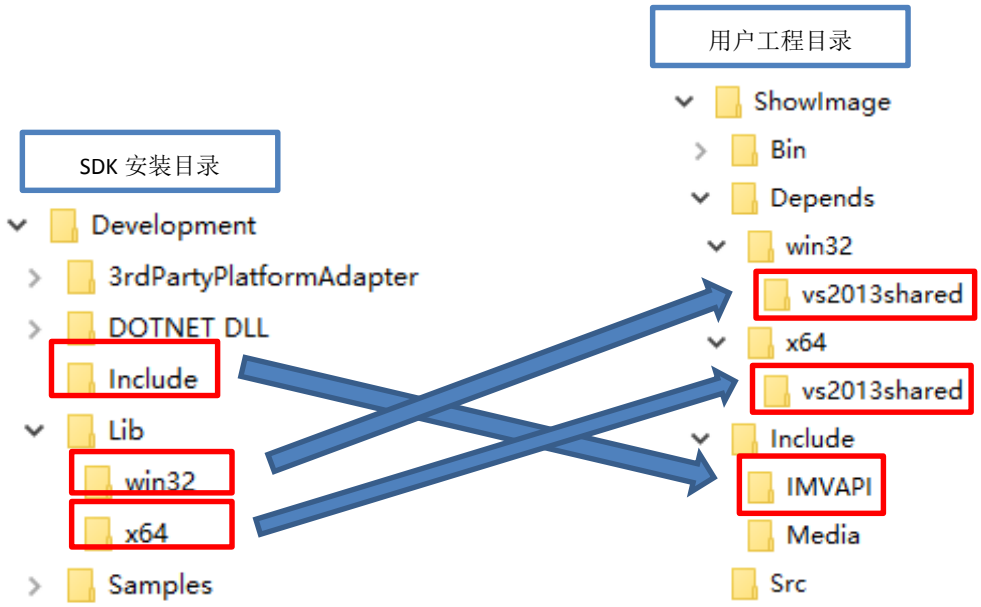


c.运行库

不需要拷贝 dll 库，默认会使用【SDK 安装目录】\Runtime 下对应平台的 dll。

1.5.3. QT（pro 文件）

①拷贝头文件



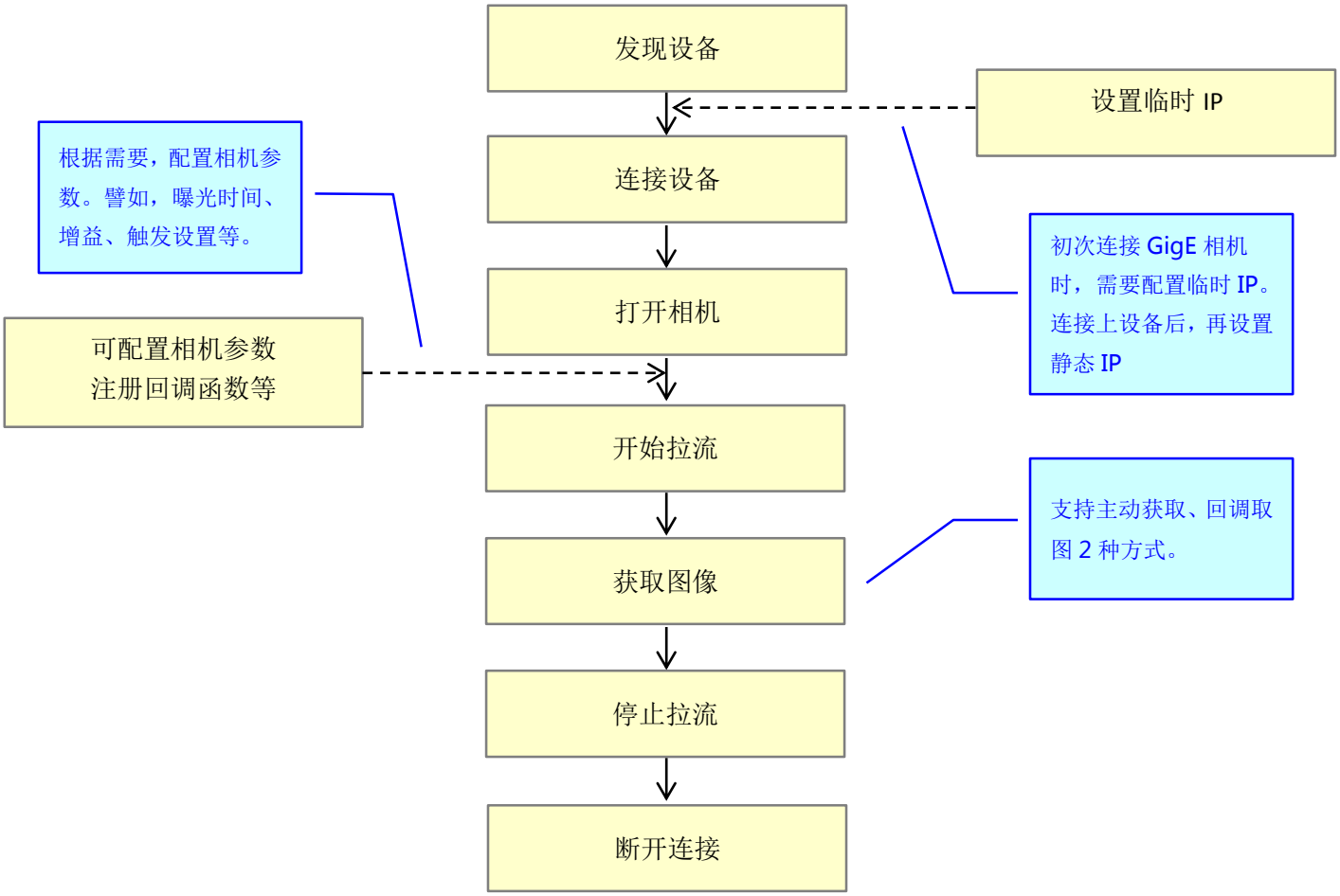
②编辑 pro 文件

```
Debug {
        contains(QMAKE_COMPILER_DEFINES, _WIN64) {
                LIBS += -L./Depends/x64/vs2013shared                -lMVSDKmd
        }
        else {
                LIBS += -L./Depends/win32/vs2013shared              -lMVSDKmd
        }
}
else {
        contains(QMAKE_COMPILER_DEFINES, _WIN64) {
                LIBS += -L./Depends/x64/vs2013shared                -lMVSDKmd
        }
        else {
                LIBS += -L./Depends/win32/vs2013shared              -lMVSDKmd
        }
}
INCLUDEPATH += ./Include
```

③运行库配置

请参照 1.4.2 VS2005 ~ 2013 的运行库配置。

2. SDK 的整体调用流程



3. SDK 的各功能接口调用说明

SDK 接口头文件路径：\Development\Include\IMVApi.h
SDK 接口的宏定义、枚举、结构体、回调函数声明路径：\Development\Include\IMVDefines.h

3.1.系统操作

3.1.1. 获取版本信息

接口：　IMV_API const char* IMV_CALL IMV_GetVersion(void);
功能说明：获取版本信息
　　成功时返回版本信息，失败时返回NULL
参考例程：暂无

3.1.2. 枚举设备（发现设备）

接口： IMV_API int IMV_CALL IMV_EnumDevices(OUT IMV_DeviceList *pDeviceList, IN unsigned int interfaceType);

功能说明：枚举设备。

pDeviceList [OUT] 设备列表

interfaceType [IN] 待枚举的接口类型, 类型可任意组合,如 interfaceTypeGige | interfaceTypeUsb3

成功，返回IMV_OK，错误，返回错误码。

1、当interfaceType = interfaceTypeAll 时，枚举所有接口下的在线设备

2、当interfaceType = interfaceTypeGige 时，枚举所有GigE网口下的在线设备

3、当interfaceType = interfaceTypeUsb3 时，枚举所有USB接口下的在线设备

4、当interfaceType = interfaceTypeCL 时，枚举所有CameraLink接口下的在线设备

5、该接口下的interfaceType支持任意接口类型的组合,如，若枚举所有GigE网口和USB3接口下的在线设备时, 可将interfaceType设置为 interfaceType = interfaceTypeGige | interfaceTypeUsb3,其它接口类型组合以此类推

参考例程：当前所有例程均有涉及

接口： IMV_API int IMV_CALL IMV_EnumDevicesByUnicast(OUT IMV_DeviceList *pDeviceList, IN const char* pIpAddress);

功能说明：以单播形式枚举设备, 仅限 Gige 设备使用。

pDeviceList [OUT] 设备列表

pIpAddress [IN] 设备的IP地址

成功，返回IMV_OK；错误，返回错误码。

参考例程：UnicastMode

3.1.3. 设备句柄的创建与销毁

接口： IMV_API int IMV_CALL IMV_CreateHandle(OUT IMV_HANDLE* handle, IN IMV_ECreateHandleMode mode, IN void* pIdentifier);

功能说明：通过指定标示符创建设备句柄，如指定索引、设备键、设备自定义名、IP地址。

Handle [OUT] 设备句柄

mode [IN] 创建设备方式

pIdentifier [IN] 指定标示符(设备键、设备自定义名、IP地址为char类型指针强转void类型指针，索引为unsigned int类型指针强转void类型指针)

成功，返回IMV_OK，错误，返回错误码。

参考例程：当前所有例程均有涉及

接口： IMV_API int IMV_CALL IMV_DestroyHandle(IN IMV_HANDLE handle);

功能说明：销毁设备句柄

handle [IN] 设备句柄

成功，返回IMV_OK；错误，返回错误码。

参考例程：当前所有例程均有涉及

3.2.相机操作

接口：　IMV_API int IMV_CALL IMV_GetDeviceInfo(IN IMV_HANDLE handle, OUT IMV_DeviceInfo *pDevInfo);
功能说明：获取设备信息
handle [IN] 设备句柄
pDevInfo [OUT] 设备信息
成功，返回IMV_OK；错误，返回错误码
参考例程：当前所有例程均有涉及

3.2.1 相机的连接与关闭

接口：　IMV_API int IMV_CALL IMV_Open(IN IMV_HANDLE handle);
功能说明：打开设备（根据句柄连接设备）
handle [IN] 设备句柄
成功，返回IMV_OK；错误，返回错误码
参考例程：当前所有例程均有涉及

接口：　IMV_API int IMV_CALL IMV_OpenEx(IN IMV_HANDLE handle, IN IMV_ECameraAccessPermission accessPermission);
功能说明：打开设备（根据句柄与访问权限连接设备）
handle [IN] 设备句柄
accessPermission [IN] 控制通道权限(IMV_Open默认使用accessPermissionControl权限)
成功，返回IMV_OK；错误，返回错误码
与IMV_Open接口的区别在于连接操作执行时需判断访问权限
参考例程：暂无

接口：　IMV_API bool IMV_CALL IMV_IsOpen(IN IMV_HANDLE handle);
功能说明：判断设备是否已打开
handle [IN] 设备句柄
打开状态，返回true；关闭状态或者掉线状态，返回false
参考例程：ForceIp

接口：　IMV_API int IMV_CALL IMV_Close(IN IMV_HANDLE handle);
功能说明：关闭设备（根据句柄断开设备）
handle [IN] 设备句柄
成功，返回IMV_OK；错误，返回错误码
参考例程：当前所有例程均有涉及

3.2.2 相机配置的下载、保存及加载配置

接口： IMV_API int IMV_CALL IMV_DownLoadGenICamXML(IN IMV_HANDLE handle, IN const char* pFullFileName);

功能说明：下载设备描述XML文件，并保存到指定路径，如：D:\\xml.zip

handle [IN] 设备句柄

pFullFileName [IN] 文件要保存的路径

成功，返回IMV_OK；错误，返回错误码

参考例程：暂无

接口： IMV_API int IMV_CALL IMV_SaveDeviceCfg(IN IMV_HANDLE handle, IN const char* pFullFileName);

功能说明：保存设备配置到指定的位置。同名文件已存在时，覆盖。

handle [IN] 设备句柄

pFullFileName [IN] 导出的设备配置文件全名(含路径)，如：D:\\config.xml 或 D:\\config.mvcfg

成功，返回IMV_OK；错误，返回错误码

参考例程：LoadAndSaveDeviceCfg

接口： IMV_API int IMV_CALL IMV_LoadDeviceCfg(IN IMV_HANDLE handle, IN const char* pFullFileName, OUT IMV_ErrorList* pErrorList);

功能说明：从文件加载设备xml配置

handle [IN] 设备句柄

pFullFileName [IN] 设备配置(xml)文件全名(含路径)，如：D:\\config.xml 或 D:\\config.mvcfg

pErrorList [OUT] 加载失败的属性名列表。存放加载失败的属性上限为 IMV_MAX_ERROR_LIST_NUM。成功，返回IMV_OK；错误，返回错误码

参考例程：LoadAndSaveDeviceCfg

3.2.3 相机配置设置（GigE 相机特有）

接口： IMV_API int IMV_CALL IMV_GIGE_ForceIpAddress(IN IMV_HANDLE handle, IN const char* pIpAddress, IN const char* pSubnetMask, IN const char* pGateway);

功能说明：修改设备IP，仅限Gige设备使用

handle [IN] 设备句柄

pIpAddress [IN] IP地址

pSubnetMask [IN] 子网掩码

pGateway [IN] 默认网关

成功，返回IMV_OK；错误，返回错误码

1、调用该函数时如果pSubnetMask和pGateway都设置了有效值，则以此有效值为准；

2、调用该函数时如果pSubnetMask和pGateway都设置了NULL，则内部实现时用它所连接网卡的子网掩码和网关代替

3、调用该函数时如果pSubnetMask和pGateway两者中其中一个为NULL，另一个非NULL，则返回错误

参考例程：ForceIp

接口： IMV_API int IMV_CALL IMV_GIGE_GetAccessPermission(IN IMV_HANDLE handle, OUT IMV_ECameraAccessPermission* pAccessPermission);

功能说明：获取设备的当前访问权限，仅限Gige设备使用

handle [IN] 设备句柄

pAccessPermission [OUT] 设备的当前访问权限

成功，返回IMV_OK；错误，返回错误码

参考例程：暂无

接口： IMV_API int IMV_CALL IMV_GIGE_SetAnswerTimeout(IN IMV_HANDLE handle, IN unsigned int timeout);

功能说明：设置设备对sdk命令的响应超时时间,仅限Gige设备使用

handle [IN] 设备句柄

timeout [IN] 超时时间，单位ms

成功，返回IMV_OK；错误，返回错误码

参考例程：GigECommunicationControl

3.2.4 本品牌相机特有功能

接口： IMV_API int IMV_CALL IMV_WriteUserPrivateData(IN IMV_HANDLE handle, IN void* pBuffer, IN_OUT unsigned int* pLength);

功能说明：写用户自定义数据。相机内部保留32768字节用于用户存储自定义数据(此功能针对本品牌相机，其它品牌相机无此功能)

handle [IN] 设备句柄

pBuffer [IN] 数据缓冲的指针

pLength [IN] 期望写入的字节数 [OUT] 实际写入的字节数

成功，返回IMV_OK；错误，返回错误码

参考例程：暂无

接口： IMV_API int IMV_CALL IMV_ReadUserPrivateData(IN IMV_HANDLE handle, OUT void* pBuffer, IN_OUT unsigned int* pLength);

功能说明：读用户自定义数据。相机内部保留32768字节用于用户存储自定义数据(此功能针对本品牌相机，其它品牌相机无此功能)

handle [IN] 设备句柄

pBuffer [OUT] 数据缓冲的指针

pLength [IN] 期望读出的字节数 [OUT] 实际读出的字节数

成功，返回IMV_OK；错误，返回错误码

参考例程：暂无

接口： IMV_API int IMV_CALL IMV_WriteUARTData(IN IMV_HANDLE handle, IN void* pBuffer, IN_OUT unsigned int* pLength);

功能说明：往相机串口寄存器写数据，每次写会清除掉上次的数据(此功能只支持包含串口功能的本品牌相机)
handle [IN] 设备句柄
pBuffer [IN] 数据缓冲的指针
pLength [IN] 期望写入的字节数 [OUT] 实际写入的字节数
成功，返回IMV_OK；错误，返回错误码
参考例程：暂无

接口： IMV_API int IMV_CALL IMV_ReadUARTData(IN IMV_HANDLE handle, OUT void* pBuffer, IN_OUT unsigned int* pLength);
功能说明：从相机串口寄存器读取串口数据(此功能只支持包含串口功能的本品牌相机)
handle [IN] 设备句柄
pBuffer [OUT] 数据缓冲的指针
pLength [IN] 期望读出的字节数 [OUT] 实际读出的字节数
成功，返回IMV_OK；错误，返回错误码
参考例程：暂无

3.3.注册事件回调

接口： IMV_API int IMV_CALL IMV_SubscribeConnectArg(IN IMV_HANDLE handle, IN IMV_ConnectCallBack proc, IN void* pUser);
功能说明：设备连接状态事件回调注册
handle [IN] 设备句柄
proc [IN] 设备连接状态事件回调函数
pUser [IN] 用户自定义数据，可设为NULL
成功，返回IMV_OK；错误，返回错误码
只支持一个回调函数,且设备关闭后，注册会失效，打开设备后需重新注册
参考例程：ResumeConnect

接口： IMV_API int IMV_CALL IMV_SubscribeParamUpdateArg(IN IMV_HANDLE handle, IN IMV_ParamUpdateCallBack proc, IN void* pUser);
功能说明：参数更新事件回调注册
handle [IN] 设备句柄
proc [IN] 参数更新注册的事件回调函数
pUser [IN] 用户自定义数据，可设为NULL
成功，返回IMV_OK；错误，返回错误码
只支持一个回调函数,且设备关闭后，注册会失效，打开设备后需重新注册
参考例程：Events

接口： IMV_API int IMV_CALL IMV_SubscribeStreamArg(IN IMV_HANDLE handle, IN IMV_StreamCallBack proc, IN void* pUser);
功能说明：流通道事件回调注册
handle [IN] 设备句柄
proc [IN] 流通道事件回调注册函数

pUser [IN] 用户自定义数据, 可设为NULL
成功，返回IMV_OK；错误，返回错误码
只支持一个回调函数,且设备关闭后，注册会失效，打开设备后需重新注册
参考例程：Events

接口： IMV_API int IMV_CALL IMV_SubscribeMsgChannelArg(IN IMV_HANDLE handle, IN IMV_MsgChannelCallBack proc, IN void* pUser);
功能说明：消息通道事件回调注册
handle [IN] 设备句柄
proc [IN] 消息通道事件回调注册函数
pUser [IN] 用户自定义数据, 可设为NULL
成功，返回IMV_OK；错误，返回错误码
只支持一个回调函数,且设备关闭后，注册会失效，打开设备后需重新注册
参考例程：Events

3.4.相机流资源操作

3.4.1. 流数据设置

接口： IMV_API int IMV_CALL IMV_SetBufferCount(IN IMV_HANDLE handle, IN unsigned int nSize);
功能说明：设置帧数据缓存个数
handle [IN] 设备句柄
nSize [IN] 缓存数量
成功，返回IMV_OK；错误，返回错误码
不能在拉流过程中设置
参考例程：暂无

接口： IMV_API int IMV_CALL IMV_ClearFrameBuffer(IN IMV_HANDLE handle);
功能说明：清除帧数据缓存
handle [IN] 设备句柄
成功，返回IMV_OK；错误，返回错误码
参考例程：ClearFrameBuffer

接口： IMV_API int IMV_CALL IMV_GIGE_SetInterPacketTimeout(IN IMV_HANDLE handle, IN unsigned int nTimeout);
功能说明：设置驱动包间隔时间(MS),仅对Gige设备有效
handle [IN] 设备句柄
nTimeout [IN] 包间隔时间，单位是毫秒
成功，返回IMV_OK；错误，返回错误码
触发模式尾包丢失重传机制
参考例程：GigECommunicationControl

接口： IMV_API int IMV_CALL IMV_GIGE_SetSingleResendMaxPacketNum(IN IMV_HANDLE handle, IN unsigned int maxPacketNum);
功能说明：设置单次重传最大包个数，仅对GigE设备有效
handle [IN] 设备句柄
maxPacketNum [IN] 单次重传最大包个数
成功，返回IMV_OK；错误，返回错误码
maxPacketNum为0时，该功能无效
参考例程：GigECommunicationControl

接口： IMV_API int IMV_CALL IMV_GIGE_SetMaxLostPacketNum(IN IMV_HANDLE handle, IN unsigned int maxLostPacketNum);
功能说明：设置同一帧最大丢包的数量,仅对GigE设备有效
handle [IN] 设备句柄
maxLostPacketNum [IN] 最大丢包的数量
成功，返回IMV_OK；错误，返回错误码
maxLostPacketNum为0时，该功能无效
只支持一个回调函数,且设备关闭后，注册会失效，打开设备后需重新注册
参考例程：GigECommunicationControl

接口： IMV_API int IMV_CALL IMV_USB_SetUrbTransfer(IN IMV_HANDLE handle, IN unsigned int nNum, IN unsigned int nSize);
功能说明：设置U3V设备的传输数据块的数量和大小,仅对USB设备有效
handle [IN] 设备句柄
nNum [IN] 传输数据块的数量(范围:5-256)
nSize [IN] 传输数据块的大小(范围:8-512, 单位:KByte)
成功，返回IMV_OK；错误，返回错误码
1、该接口暂时只有在Linux平台且使用无驱的情况下设置才有效，其他情况下返回IMV_NOT_SUPPORT错误码
2、传输数据块数量，范围5 - 256, 默认为64，该值设置过小会影响传输效率；多台相机共同使用时，可以适当减小该值
3、传输每个数据块大小，范围8 - 512, 默认为64，单位是KByte
参考例程：暂无

3.4.2. 图像采集

接口： IMV_API int IMV_CALL IMV_StartGrabbing(IN IMV_HANDLE handle);
功能说明：开始取流
handle [IN] 设备句柄
成功，返回IMV_OK；错误，返回错误码
参考例程：Grab

接口： IMV_API int IMV_CALL IMV_StartGrabbingEx(IN IMV_HANDLE handle, IN uint64_t maxImagesGrabbed, IN IMV_EGrabStrategy strategy);
功能说明：开始取流

handle [IN] 设备句柄
maxImagesGrabbed [IN] 允许最多的取帧数，达到指定取帧数后停止取流，如果为0，表示忽略此参数连续取流(IMV_StartGrabbing默认0)
strategy [IN] 取流策略,(IMV_StartGrabbing默认使用grabStrartegySequential策略取流)
成功，返回IMV_OK；错误，返回错误码
参考例程：GrabStrategy、GrabStrategyUpcomingImage

接口： IMV_API bool IMV_CALL IMV_IsGrabbing(IN IMV_HANDLE handle);
功能说明：判断设备是否正在取流
handle [IN] 设备句柄
正在取流，返回true；不在取流，返回false
参考例程：MultipleCamera

接口： IMV_API int IMV_CALL IMV_StopGrabbing(IN IMV_HANDLE handle);
功能说明：停止取流
handle [IN] 设备句柄
成功，返回IMV_OK；错误，返回错误码
参考例程：Grab

3.4.3. 获取图像数据

接口： IMV_API int IMV_CALL IMV_AttachGrabbing(IN IMV_HANDLE handle, IN IMV_FrameCallBack proc, IN void* pUser);
功能说明：注册帧数据回调函数(异步获取帧数据机制)
handle [IN] 设备句柄
proc [IN] 帧数据信息回调函数，建议不要在该函数中处理耗时的操作，否则会阻塞后续帧数据的实时性
pUser [IN] 用户自定义数据，可设为NULL
成功，返回IMV_OK；错误，返回错误码
该异步获取帧数据机制和同步获取帧数据机制(IMV_GetFrame)互斥，对于同一设备，系统中两者只能选其一
只支持一个回调函数，且设备关闭后，注册会失效，打开设备后需重新注册
参考例程：ChunkData、Events、FrameClone、FrameTriggerCount、GrabCallback

接口： IMV_API int IMV_CALL IMV_GetFrame(IN IMV_HANDLE handle, OUT IMV_Frame* pFrame, IN unsigned int timeoutMS);
功能说明：获取一帧图像(同步获取帧数据机制)
handle [IN] 设备句柄
pFrame [OUT] 帧数据信息
timeoutMS [IN] 获取一帧图像的超时时间,INFINITE时表示无限等待,直到收到一帧数据或者停止取流。单位是毫秒
成功，返回IMV_OK；错误，返回错误码

该接口不支持多线程调用。
该同步获取帧机制和异步获取帧机制(IMV_AttachGrabbing)互斥,对于同一设备，系统中两者只能选其一。
使用内部缓存获取图像，需要IMV_ReleaseFrame进行释放图像缓存。
参考例程：ClearFrameBuffer、Grab

接口： IMV_API int IMV_CALL IMV_ReleaseFrame(IN IMV_HANDLE handle, IN IMV_Frame* pFrame);
功能说明：释放图像缓存
handle [IN] 设备句柄
pFrame [IN] 帧数据信息
成功，返回IMV_OK；错误，返回错误码
参考例程：ClearFrameBuffer、Grab

3.4.4. 流与图像数据的其他操作

接口： IMV_API int IMV_CALL IMV_CloneFrame(IN IMV_HANDLE handle, IN IMV_Frame* pFrame, OUT IMV_Frame* pCloneFrame);
功能说明：帧数据深拷贝克隆
handle [IN] 设备句柄
pFrame [IN] 克隆源帧数据信息
pCloneFrame [OUT] 新的帧数据信息
成功，返回IMV_OK；错误，返回错误码
使用IMV_ReleaseFrame进行释放图像缓存。
参考例程：FrameClone

接口： IMV_API int IMV_CALL IMV_GetChunkDataByIndex(IN IMV_HANDLE handle, IN IMV_Frame* pFrame, IN unsigned int index, OUT IMV_ChunkDataInfo *pChunkDataInfo);
功能说明：获取Chunk数据(仅对GigE/Usb相机有效)
handle [IN] 设备句柄
pFrame [IN] 帧数据信息
index [IN] 索引ID
pChunkDataInfo [OUT] Chunk数据信息
成功，返回IMV_OK；错误，返回错误码
参考例程：ChunkData

接口： IMV_API int IMV_CALL IMV_GetStatisticsInfo(IN IMV_HANDLE handle, OUT IMV_StreamStatisticsInfo* pStreamStatsInfo);
功能说明：获取流统计信息(IMV_StartGrabbing / IMV_StartGrabbing执行后调用)
handle [IN] 设备句柄
pStreamStatsInfo [OUT] 流统计信息数据
成功，返回IMV_OK；错误，返回错误码
参考例程：StreamStatistics

接口： IMV_API int IMV_CALL IMV_ResetStatisticsInfo(IN IMV_HANDLE handle);
功能说明：重置流统计信息(IMV_StartGrabbing / IMV_StartGrabbing执行后调用)
handle [IN] 设备句柄
成功，返回IMV_OK；错误，返回错误码
参考例程：StreamStatistics

3.5.属性操作

3.5.1. 属性的判断、获取、设置

① 属性的判断
接口： IMV_API bool IMV_CALL IMV_FeatureIsAvailable(IN IMV_HANDLE handle, IN const char* pFeatureName);
功能说明：判断属性是否可用
handle [IN] 设备句柄
pFeatureName [IN] 属性名
可，返回true；不可，返回false
参考例程：暂无

接口： IMV_API bool IMV_CALL IMV_FeatureIsReadable (IN IMV_HANDLE handle, IN const char* pFeatureName);
功能说明：判断属性是否可读
handle [IN] 设备句柄
pFeatureName [IN] 属性名
可，返回true；不可，返回false
参考例程：暂无

接口： IMV_API bool IMV_CALL IMV_FeatureIsWriteable (IN IMV_HANDLE handle, IN const char* pFeatureName);
功能说明：判断属性是否可写
handle [IN] 设备句柄
pFeatureName [IN] 属性名
可，返回true；不可，返回false
参考例程：暂无

接口： IMV_API bool IMV_CALL IMV_FeatureIsStreamable (IN IMV_HANDLE handle, IN const char* pFeatureName);
功能说明：判断属性是否可流
handle [IN] 设备句柄
pFeatureName [IN] 属性名
可，返回true；不可，返回false
参考例程：暂无

接口: IMV_API bool IMV_CALL IMV_FeatureIsValid (IN IMV_HANDLE handle, IN const char* pFeatureName);
功能说明:判断属性是否有效
handle [IN] 设备句柄
pFeatureName [IN] 属性名
可,返回true;不可,返回false
参考例程:Record

② 属性的获取
接口: IMV_API int IMV_CALL IMV_GetIntFeatureValue(IN IMV_HANDLE handle, IN const char* pFeatureName, OUT int64_t* pIntValue);
功能说明:获取整型属性值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pIntValue [OUT] 整型属性值
成功,返回IMV_OK;错误,返回错误码
参考例程:CommPropAccess

接口: IMV_API int IMV_CALL IMV_GetIntFeatureMin (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT int64_t* pIntValue);
功能说明:获取整型属性可设的最小值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pIntValue [OUT] 整型属性可设的最小值
成功,返回IMV_OK;错误,返回错误码
参考例程:CommPropAccess

接口: IMV_API int IMV_CALL IMV_GetIntFeatureMax (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT int64_t* pIntValue);
功能说明:获取整型属性可设的最大值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pIntValue [OUT] 整型属性可设的最大值
成功,返回IMV_OK;错误,返回错误码
参考例程:CommPropAccess

接口: IMV_API int IMV_CALL IMV_GetIntFeatureInc (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT int64_t* pIntValue);
功能说明:获取整型属性步长
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pIntValue [OUT] 整型属性步长
成功,返回IMV_OK;错误,返回错误码
参考例程:CommPropAccess

接口： IMV_API int IMV_CALL IMV_GetDoubleFeatureValue (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT double* pDoubleValue);
功能说明：获取浮点属性值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pIntValue [OUT] 浮点属性值
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess

接口： IMV_API int IMV_CALL IMV_GetDoubleFeatureMin (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT double* pDoubleValue);
功能说明：获取浮点属性可设的最小值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pIntValue [OUT] 浮点属性可设的最小值
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess

接口： IMV_API int IMV_CALL IMV_GetDoubleFeatureMax (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT double* pDoubleValue);
功能说明：获取浮点属性可设的最大值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pIntValue [OUT] 浮点属性可设的最大值
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess

接口： IMV_API int IMV_CALL IMV_GetBoolFeatureValue (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT bool* pBoolValue);
功能说明：获取布尔属性值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pIntValue [OUT] 布尔属性值
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess

接口： IMV_API int IMV_CALL IMV_GetStringFeatureValue (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT IMV_String* pStringValue);
功能说明：获取字符串属性值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pIntValue [OUT] 字符串属性值
成功，返回IMV_OK；错误，返回错误码

参考例程：CommPropAccess

接口：　IMV_API int IMV_CALL IMV_GetEnumFeatureValue (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT uint64_t* pEnumValue);
功能说明：获取枚举属性值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pEnumValue [OUT] 枚举属性值
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess

接口：　IMV_API int IMV_CALL IMV_GetEnumFeatureSymbol (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT IMV_String* pEnumValueSymbol);
功能说明：获取枚举属性symbol值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pEnumValue [OUT] 枚举属性symbol值
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess
参考例程：暂无

接口：　IMV_API int IMV_CALL IMV_GetEnumFeatureEntryNum (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT unsigned int* pEntryNum);
功能说明：获取枚举属性值可设置个数
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pEntryNum [OUT] 枚举属性值可设置个数
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess

接口：　IMV_API int IMV_CALL IMV_GetEnumFeatureEntrys (IN IMV_HANDLE handle, IN const char* pFeatureName, OUT IMV_EnumEntryList* pEnumEntryList);
功能说明：获取枚举属性值可设置枚举值列表
handle [IN] 设备句柄
pFeatureName [IN] 属性名
pEnumEntryList [OUT] 枚举属性值可设置枚举值列表
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess

③　属性的设置

接口：　IMV_API int IMV_CALL IMV_SetIntFeatureValue(IN IMV_HANDLE handle, IN const char* pFeatureName, IN int64_t intValue);
功能说明：设置整型属性值
handle [IN] 设备句柄

pFeatureName [IN] 属性名
intValue [IN] 待设置的整型属性值
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess

接口： IMV_API int IMV_CALL IMV_SetDoubleFeatureValue (IN IMV_HANDLE handle, IN const char* pFeatureName, IN double doubleValue);
功能说明：设置浮点属性值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
doubleValue [IN] 待设置的浮点属性值
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess

接口： IMV_API int IMV_CALL IMV_SetBoolFeatureValue (IN IMV_HANDLE handle, IN const char* pFeatureName, IN bool boolValue);
功能说明：设置布尔属性值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
boolValue [IN] 待设置的布尔属性值
成功，返回IMV_OK；错误，返回错误码
参考例程：ChunkData

接口： IMV_API int IMV_CALL IMV_SetEnumFeatureValue (IN IMV_HANDLE handle, IN const char* pFeatureName, IN uint64_t enumValue);
功能说明：设置枚举属性值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
enumValue [IN] 待设置的枚举属性值
成功，返回IMV_OK；错误，返回错误码
参考例程：CommPropAccess

接口： IMV_API int IMV_CALL IMV_SetEnumFeatureSymbol (IN IMV_HANDLE handle, IN const char* pFeatureName, IN const char* pEnumSymbol);
功能说明：设置枚举属性symbol值
handle [IN] 设备句柄
pFeatureName [IN] 属性名
boolValue [IN] 待设置的枚举属性symbol值
成功，返回IMV_OK；错误，返回错误码
参考例程：ChunkData、CommPropAccess、SoftTrigger、UserSetControl

接口： IMV_API int IMV_CALL IMV_SetStringFeatureValue (IN IMV_HANDLE handle, IN const char* pFeatureName, IN const char* pStringValue);
功能说明：设置字符串属性值

handle [IN] 设备句柄
pFeatureName [IN] 属性名
pStringValue [IN] 待设置的字符串属性值
成功，返回IMV_OK；错误，返回错误码

参考例程：CommPropAccess

3.5.2. 命令属性的执行

接口： `IMV_API int IMV_CALL IMV_ExecuteCommandFeature(IN IMV_HANDLE handle, IN const char* pFeatureName);`

功能说明： 执行命令属性
handle [IN] 设备句柄
pFeatureName [IN] 属性名
成功，返回IMV_OK；错误，返回错误码

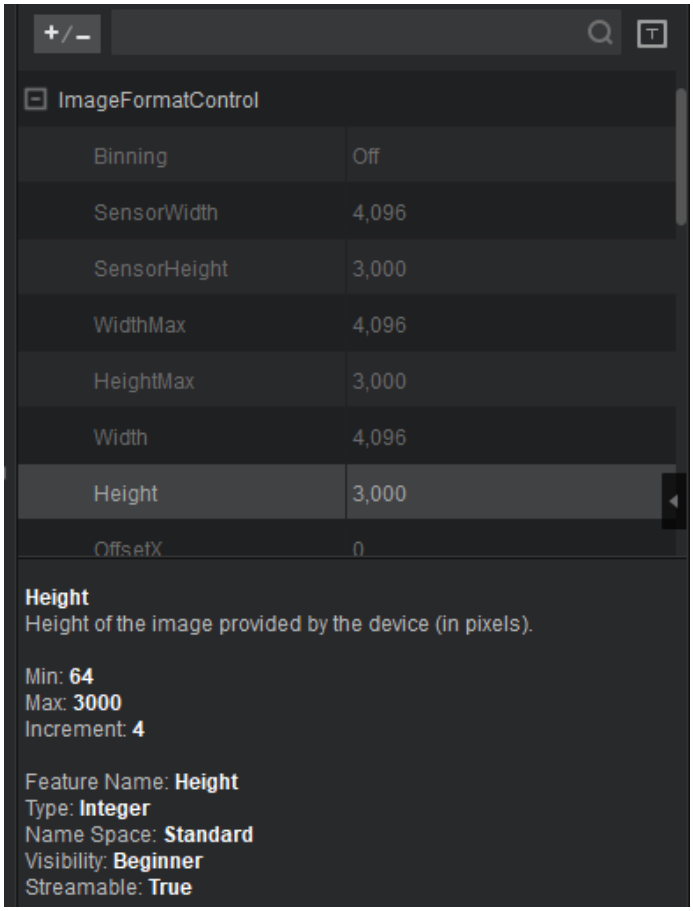
参考例程： ChunkData、ClearFrameBuffer、 SoftTrigger、UserSetControl

3.5.3. 属性的查询与使用

打开相机客户端可查询对应的属性，并且可以了解属性的详细信息，如：数据类型、范围等。

1) 整型属性

例：若需获取相机的像素高度信息，找到 Height 属性并选中，下方会显示该属性的数据类型，根据 Type 的显示而定，下图 Height 属性的数据类型为 int 整型。

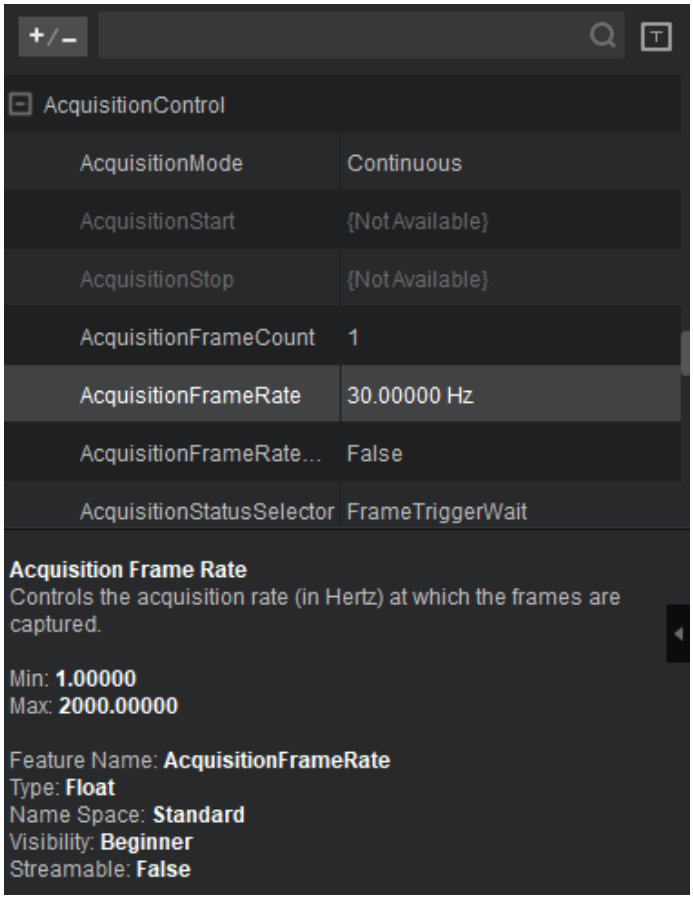


随后对所找到属性进行获取及修改。

```
int ret = IMV_OK;
int64_t heightValue = 0;
int64_t heightMinValue = 0;
int64_t heightMaxValue = 0;
int64_t incrementValue = 0;

// 获取属性值
ret = IMV_GetIntFeatureValue(devHandle, "Height", &heightValue);
if (IMV_OK != ret)
{
    printf("Get feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}

// 获取属性可设的最小值，若无最小值则此操作错误
ret = IMV_GetIntFeatureMin(devHandle, "Height", &heightMinValue);
if (IMV_OK != ret)
{
    printf("Get feature minimum value failed! ErrorCode[%d]\n", ret);
    return ret;
}

// 获取属性可设的最大值，若无最大值则此操作错误
ret = IMV_GetIntFeatureMax(devHandle, "Height", &heightMaxValue);
if (IMV_OK != ret)
{
    printf("Get feature maximum value failed! ErrorCode[%d]\n", ret);
    return ret;
}

// 获取属性步长，若无步长值则此操作错误
ret = IMV_GetIntFeatureInc(devHandle, "Height", &incrementValue);
if (IMV_OK != ret)
{
    printf("Get feature increment value failed! ErrorCode[%d]\n", ret);
    return ret;
}

// 设置属性值
ret = IMV_SetIntFeatureValue(devHandle, "Height", heightValue);
if (IMV_OK != ret)
{
    printf("Set feature value failed! ErrorCode[%d]\n", ret);
```

```
    return ret;
}
```

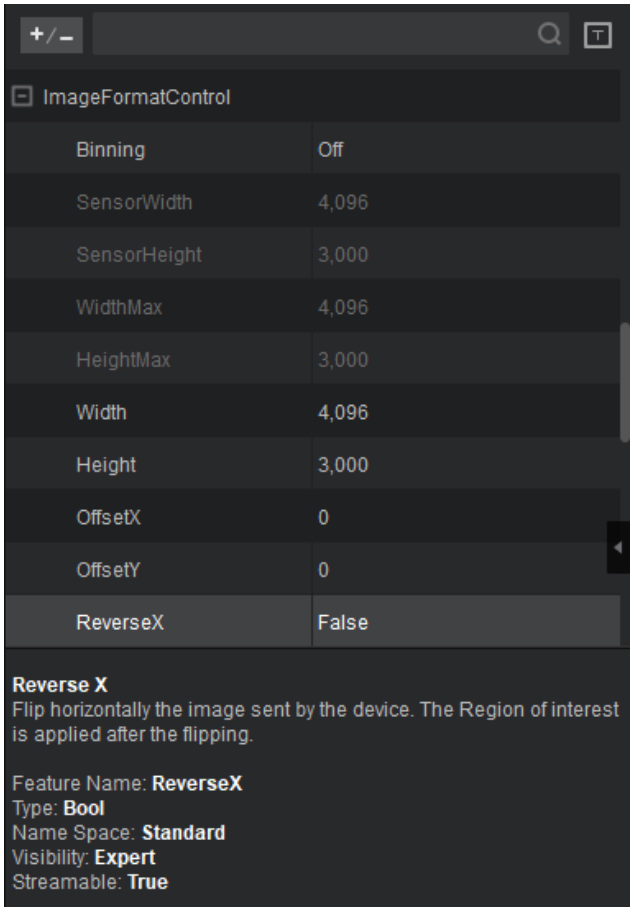
2) 浮点型属性




```
int ret = IMV_OK;
float framerate = 0;
float frameratemin = 0;
float frameratemax = 0;

// 获取属性值
ret = IMV_GetDoubleFeatureValue (devHandle, "AcquisitionFrameRate", &framerate);
if (IMV_OK != ret)
{
    printf("Get feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}
// 获取属性可设的最小值，若无最小值则此操作错误
ret = IMV_GetDoubleFeatureMin(devHandle, "AcquisitionFrameRate", &frameratemin);
if (IMV_OK != ret)
{
    printf("Get feature minimum value failed! ErrorCode[%d]\n", ret);
    return ret;
}
```

```
// 获取属性可设的最大值，若无最大值则此操作错误
ret = IMV_GetDoubleFeatureMax(devHandle, "AcquisitionFrameRate", &frameratemax);
if (IMV_OK != ret)
{
    printf("Get feature maximum value failed! ErrorCode[%d]\n", ret);
    return ret;
}

// 设置属性值
ret = IMV_SetDoubleFeatureValue(devHandle, "AcquisitionFrameRate", framerate);
if (IMV_OK != ret)
{
    printf("Set feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}
```

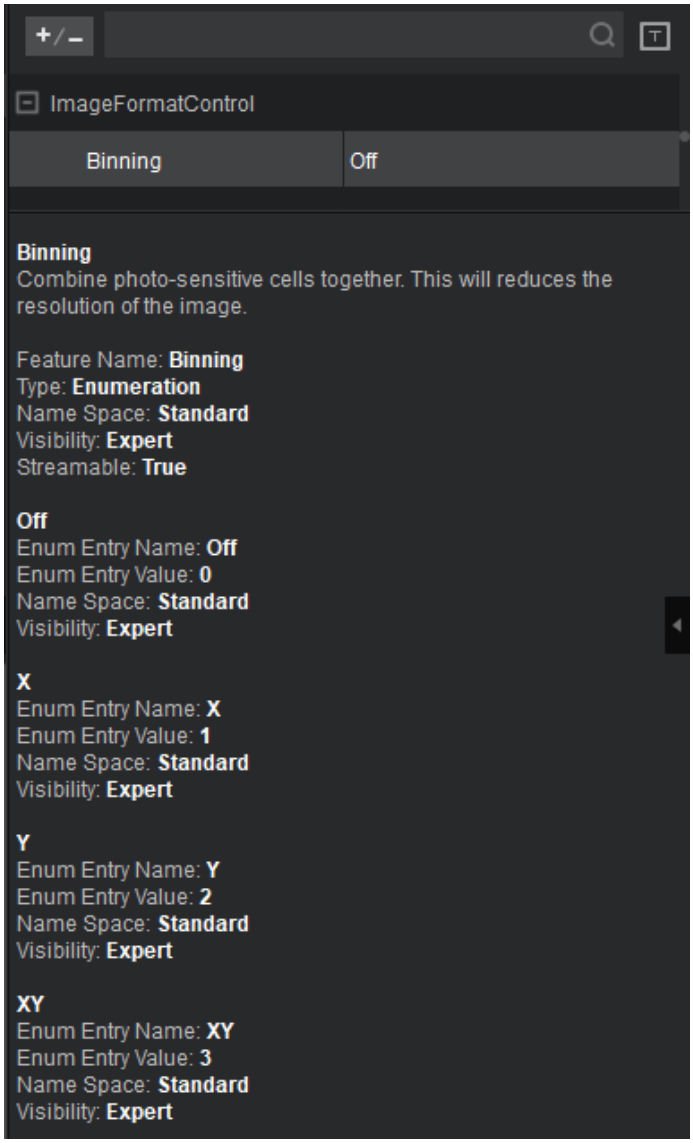
3) 布尔属性




```
int ret = IMV_OK;
bool reversevalue = 0;

// 获取属性值
ret = IMV_GetBoolFeatureValue (devHandle, "ReverseX", &reversevalue);
```

```
if (IMV_OK != ret)
{
    printf("Get feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}

// 设置属性值
ret = IMV_SetBoolFeatureValue(devHandle, "ReverseX", reversevalue);
if (IMV_OK != ret)
{
    printf("Set feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}
```

4) 枚举属性

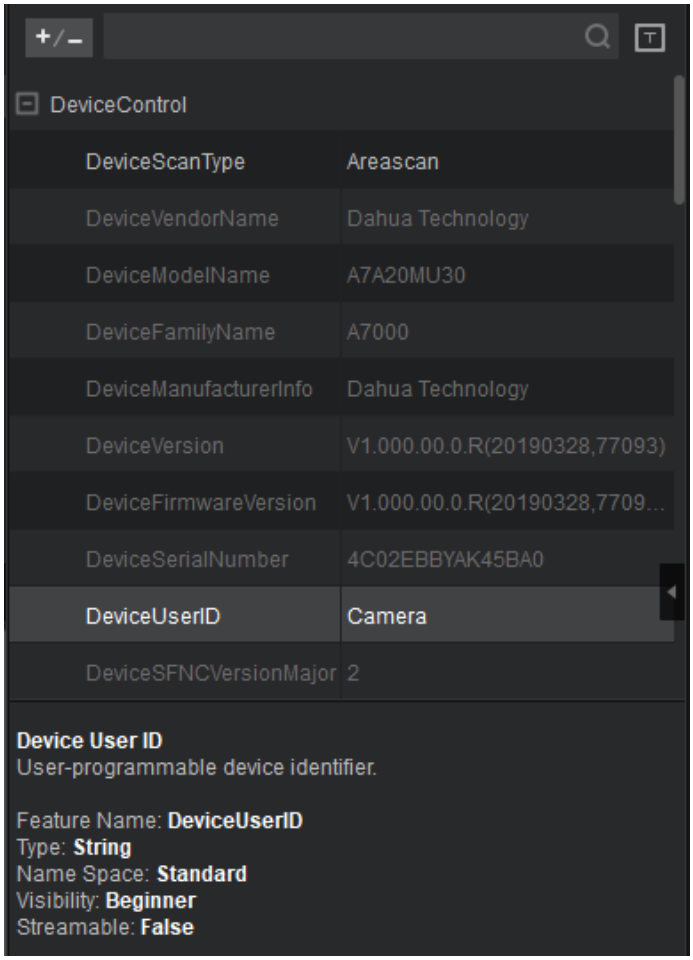



```
int ret = IMV_OK;
uint64_t enumValue;
```

```
IMV_String pEnumSymbol;
unsigned int pEntryNum;
IMV_EnumEntryList pEnumEntryList;

// 获取属性值的可设枚举值列表
ret = IMV_GetEnumFeatureValue(devHandle, "Binning", &pEnumEntryList);
if (IMV_OK != ret)
{
    printf("Get feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}

// 获取属性可设枚举值的个数
ret = IMV_GetEnumFeatureEntryNum(devHandle, "Binning", &pEntryNum);
if (IMV_OK != ret)
{
    printf("Get feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}

// 获取属性symbol值
ret = IMV_GetEnumFeatureSymbol(devHandle, "Binning", &pEnumSymbol);
if (IMV_OK != ret)
{
    printf("Get feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}

// 获取属性值
ret = IMV_GetIntFeatureInc(devHandle, "Binning", &enumValue);
if (IMV_OK != ret)
{
    printf("Get feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}

// 设置属性symbol值
ret = IMV_SetEnumFeatureValue(devHandle, "Binning", pEnumSymbol);
if (IMV_OK != ret)
{
    printf("Set feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}
```

```
// 设置属性值
    ret = IMV_SetEnumFeatureValue(devHandle, "Binning", enumValue);
    if (IMV_OK != ret)
    {
        printf("Set feature value failed! ErrorCode[%d]\n", ret);
        return ret;
    }
```

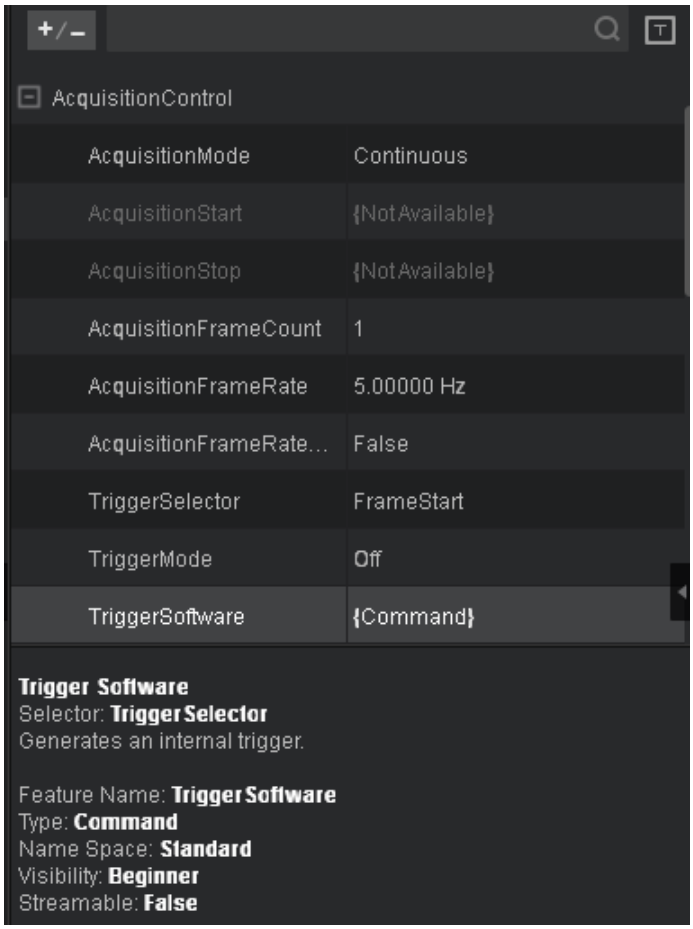
5) 字符串属性




```
    int ret = IMV_OK;
    string id = "";

    // 获取属性值
    ret = IMV_GetStringFeatureValue (devHandle, "DeviceUserID", &id);
    if (IMV_OK != ret)
    {
        printf("Get feature value failed! ErrorCode[%d]\n", ret);
        return ret;
    }

    // 设置属性值
```

```
ret = IMV_SetStringFeatureValue(devHandle, "DeviceUserID", id);
if (IMV_OK != ret)
{
    printf("Set feature value failed! ErrorCode[%d]\n", ret);
    return ret;
}
```

6) 命令属性




```
int ret = IMV_OK;
// 执行命令属性值
ret = IMV_ExecuteCommandFeature (devHandle, "TriggerSoftware");
if (IMV_OK != ret)
{
    printf("Execute Command feature failed! ErrorCode[%d]\n", ret);
    return ret;
}
```

3.6 图像操作

3.6.1 录像相关

接口： IMV_API int IMV_CALL IMV_OpenRecord(IN IMV_HANDLE handle, IN IMV_RecordParam *pstRecordParam);
功能说明：打开录像
handle [IN] 设备句柄
pstRecordParam [IN] 录像参数结构体
成功，返回IMV_OK；错误，返回错误码
参考例程：Record

接口： IMV_API int IMV_CALL IMV_InputOneFrame(IN IMV_HANDLE handle, IN IMV_RecordFrameInfoParam *pstRecordFrameInfoParam);
功能说明：录制一帧图像
handle [IN] 设备句柄
pstRecordFrameInfoParam [IN] 录像用帧信息结构体
成功，返回IMV_OK；错误，返回错误码
参考例程：暂无

接口： IMV_API int IMV_CALL IMV_CloseRecord(IN IMV_HANDLE handle);
功能说明：关闭录像
handle [IN] 设备句柄
成功，返回IMV_OK；错误，返回错误码
参考例程：Record

3.6.2 图像相关

接口： IMV_API int IMV_CALL IMV_PixelConvert(IN IMV_HANDLE handle, IN_OUT IMV_PixelConvertParam* pstPixelConvertParam);
功能说明：像素格式转换
handle [IN] 设备句柄
pstPixelConvertParam [IN][OUT] 像素格式转换参数结构体
成功，返回IMV_OK；错误，返回错误码
只支持转化成目标像素格式gvspPixelRGB8 / gvspPixelBGR8 / gvspPixelMono8 / gvspPixelBGRA8
通过该接口将原始图像数据转换成用户所需的像素格式并存放在调用者指定内存中。
像素格式为YUV411Packed的时，图像宽须能被4整除
像素格式为YUV422Packed的时，图像宽须能被2整除
像素格式为YUYVPacked的时，图像宽须能被2整除
转换后的图像:数据存储是从最上面第一行开始的，这个是相机数据的默认存储方向
参考例程：ImageConvert、SaveImageToBmp

接口: IMV_API int IMV_CALL IMV_FlipImage(IN IMV_HANDLE handle, IN_OUT IMV_FlipImageParam* pstFlipImageParam);

功能说明: 图像翻转

handle [IN] 设备句柄

pstFlipImageParam [IN][OUT] 图像翻转参数结构体

成功，返回IMV_OK；错误，返回错误码

只支持像素格式gvspPixelRGB8 / gvspPixelBGR8 / gvspPixelMono8的图像的垂直和水平翻转。

通过该接口将原始图像数据翻转后并存放在调用者指定内存中。

参考例程: ImageFlip

接口: IMV_API int IMV_CALL IMV_RotateImage(IN IMV_HANDLE handle, IN_OUT IMV_RotateImageParam* pstRotateImageParam);

功能说明: 图像顺时针旋转

handle [IN] 设备句柄

pstRotateImageParam [IN][OUT] 图像旋转参数结构体

成功，返回IMV_OK；错误，返回错误码

只支持gvspPixelRGB8 / gvspPixelBGR8 / gvspPixelMono8格式数据的90/180/270度顺时针旋转。

通过该接口将原始图像数据旋转后并存放在调用者指定内存中。

参考例程: ImageRotate

4. 第三方平台的图像对象转换方法

4.1. Halcon 对象

4.1.1. 相机图像格式为 Mono8 时（主要是黑白格式）

Mono8 不需要转码，可以直接转换为 Halcon 图像对象。

这里假设 frame 是 SDK 取流得到的 IMV_Frame 数据帧结构体。

```
Hobject grabImg; //Halcon 图像对象
gen_image1_extern(&(grabImg),
                  "byte",
                  (Hlong) frame.frameInfo.width,
                  (Hlong) frame.frameInfo.height,
                  (Hlong)( frame.pData ),
                  0);
```

4.1.2. 相机图像格式为 Mono8 以外时（主要是彩色格式）

非 Mono8 格式需要先转换为 BGR。

图像转码请参考 3.5.4 的图像格式转换。

这里假设 frame 是 SDK 取流得到的 IMV_Frame 数据帧结构体，pBGRbuffer 是转码之后的 BGR 数据。

```
Hobject grabImg; //Halcon 图像对象
gen_image_interleaved(&grabImg,
                      (Hlong) pBGRbuffer,
                      "bgr", frame.frameInfo.width,
                      frame.frameInfo.height, 0,
                      "byte", frame.frameInfo.width,
                      frame.frameInfo.height,0,0,8,0);
```

4.2. OpenCV 对象

4.2.1. 相机图像格式为 Mono8 时（主要是黑白格式）

Mono8 不需要转码，可以直接转换为 Halcon 图像对象。

这里假设 frame 是 SDK 取流得到的 IMV_Frame 数据帧结构体。

```
//转换为 cv::Mat 对象
cv::Mat image = cv::Mat(frame.frameInfo.height,
                        frame.frameInfo.width,
                        CV_8U,
                        (uint8_t*)(( frame.pData));
```

4.2.2. 相机图像格式为 Mono8 以外时（主要是彩色格式）

非 Mono8 格式需要先转换为 BGR。

图像转码请参考 3.5.4 的图像格式转换。

这里假设 frame 是 SDK 取流得到的 IMV_Frame 数据帧结构体，pBGRbuffer 是转码之后的 BGR 数据。

```
//转换为 cv::Mat 对象
Mat image = cv::Mat(frame.frameInfo.height,
                    frame.frameInfo.width,
                    CV_8UC3,
                    (uint8_t*)pBGRbuffer);
```

4.3. QT 对象

4.3.1. 相机图像格式为 Mono8 时（主要是黑白格式）

Mono8 不需要转码，可以直接转换为 Halcon 图像对象。
这里假设 frame 是 SDK 取流得到的 IMV_Frame 数据帧结构体。

```
QImage image; //Qt 的 Qimage 对象
image = QImage(frame.pData,
                frame.frameInfo.width,
                frame.frameInfo.height,
                QImage::Format_Grayscale8);
```

4.3.2. 相机图像格式为 Mono8 以外时（主要是彩色格式）

非 Mono8 格式需要先转换为 BGR。
图像转码请参考 3.5.4 的图像格式转换。
这里假设 frame 是 SDK 取流得到的 IMV_Frame 数据帧结构体，pBGRbuffer 是转码之后的 BGR 数据。

```
QImage image; //Qt 的 Qimage 对象
image = QImage((uint8_t*) pBGRbuffer,
                frame.frameInfo.width,
                frame.frameInfo.height,
                QImage::Format_RGB888);
```

5. 配套例程说明

C 例程路径：【SDK 安装目录】\Development\Samples\VC 。

例程名	例程说明
ChunkData	获取块数据
ClearFrameBuffer	清除图像缓存
CommPropAccess	通用属性使用例程
ConnectSpecCamera	连接指定的相机
DynamicallyLoadDLL	动态加载库
Events	事件处理机制
ForceIp	修改相机 IP-仅限 GigE 相机需要
FrameClone	帧克隆
FrameTriggerCount	帧触发统计
GigECommunicationControl	GigE 通信控制-仅限 GigE 相机需要

Grab	自由模式采图（主动取流）
GrabCallback	拉流回调
GrabStrategy	拉流策略
GrabStrategyUpcomingImage	拉流处理
ImageConvert	图像转化
ImageFlip	图像翻转
ImageRotate	图像顺时针旋转
LineTrigger	外触发方式采图（回调取流）
LoadAndSaveDeviceCfg	保存和加载相机属性配置文件
MultipleCamera	多相机拉流
Record	录像
ResumeConnect	断线重连
SaveImageToBmp	保存 BMP 图像
SoftTrigger	软触发方式采图（回调取流）
StreamStatistics	流统计
UnicastMode	单播发现设备-仅限 GigE 设备
UserSetControl	保存和加载相机配置

6. 常见问题&注意点

6.1. 客户的程序使用 C 接口采图，只能取到 8 张图像（和 SDK 缓存个数一致）。

原因： C 接口取到图像对象后，没有释放，导致缓存被占满，无法再取到流。

解决方法：如下图所示，每取到帧 frame 都需要调用 IMV_ReleaseFrame 接口进行释放。

```
// 获取一帧图像
// Get a frame image
ret = IMV_GetFrame(devHandle, &frame, 1000);
if (IMV_OK != ret)
{
    printf("Get frame failed! ErrorCode[%d]\n", ret);
    return ret;
}

printf("Get frame blockId = %llu\n", frame.frameInfo.blockId);

// 释放图像缓存
// Free image buffer
ret = IMV_ReleaseFrame(devHandle, &frame);
if (IMV_OK != ret)
{
    printf("Release frame failed! ErrorCode[%d]\n", ret);
    return ret;
}
```

- END -