视沃科技

大牛直播 SDK

Windows 平台 SDK 集成说明

官网：https://daniusdk.com

Github：https://github.com/daniulive/SmarterStreaming

声　明

非常感谢您选用我们的 SDK，您的支持将激励我们持续进步。

随着产品的迭代，产品手册会在每个版本发布后不定期更新，最新版本请以官方网站(https://daniusdk.com)为准。

本手册中内容仅为开发者提供参考指导作用，具体调用请以 SDK 示例为准。

目 录

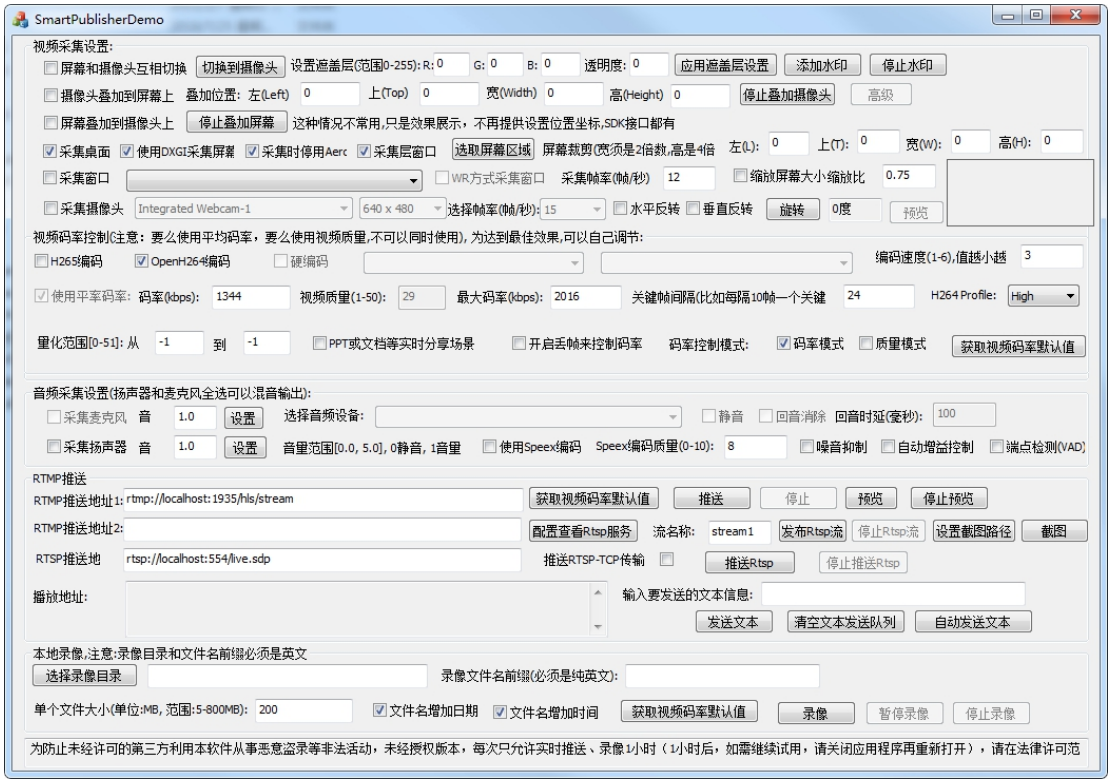
1. Windows 平台 RTMP/RTSP 直播推送 SDK

- 大牛直播 SDK 自有框架，易于扩展，自适应算法让延迟更低、采集编码传输效率更高；
- 所有功能以 SDK 接口形式提供，所有状态，均有 event 回调，完美支持断网自动重连；
- SDK 模块化，可和大牛直播播放器 SDK 组合实现流媒体数据转发、连麦、一对一互动等场景；
- 推送叠加以层级模式提供，开发者可以自行组合数据源(如多摄像头/屏幕/水印叠加)；
- 支持外部 YUV/RGB/H.264/AAC/SPEEX/PCMA/PCMU 数据源接入；
- 所有参数均可通过 SDK 接口单独设置，亦可通过默认参数，傻瓜式设置；
- 推送、录像、内置轻量级 RTSP 服务模块完全分离，可单独使用亦可组合使用；
- 业内甚至很难找到效果接近的 SDK。

1.1 demo 说明

- 大牛直播 SDK 提供 C++/C#两套接口，对外提供 32/64 位 debug/release 库，C++和 C#接口一一对应，C#接口比 C++接口增加前缀 NT_PB_。
- WIN-PublisherSDK-CPP-Demo：推送端 SDK 对应的 C++接口的 demo；
- WIN-PublisherSDK-CSharp-Demo：推送端 SDK 对应的 C#接口的 demo；
- 推送端 SDK 支持 Win7 及以上系统。
- 本 demo 基于 VS2013 开发。

1.2 界面 UI 展示



1.3 集成说明

C++头文件：

- [类型定义]nt_type_define.h
- [Log 定义]smart_log.h
- [Log 定义]smart_log_define.h
- [音视频类型定义]nt_common_media_define.h
- [base code 定义]nt_base_code_define.h
- [publisher 接口]nt_smart_publisher_define.h
- [publisher 接口]nt_smart_publisher_sdk.h

C#头文件：

- [Log 定义]smart_log.cs
- [Log 定义]smart_log_define.cs
- [音视频类型定义]nt_common_media_define.cs
- [base code 定义]nt_base_code_define.cs
- [publisher 接口]nt_smart_publisher_define.cs
- [publisher 参数定义]nt_smart_publisher_sdk.cs

相关 Lib：

- **SmartLog.dll**
- **SmartLog.lib**
- **SmartPublisherSDK.dll**
- **SmartPublisherSDK.lib**
- **NTSmartPublisherWinRTSDK.dll**
- avcodec-56.dll
- avdevice-56.dll
- avfilter-5.dll
- avformat-56.dll
- avutil-54.dll
- postproc-53.dll
- swresample-1.dll
- swscale-3.dll

集成步骤：

1. 把 lib 目录下 debug/release 库拷贝到需要集成的工程对应的 debug 或 release 目录下（确保 32 位/64 位库 debug/release 目录一一对应）；

   lib 目录如下：

   (1) 32 位 debug 库：debug

   (2) 32 位 release 库：release

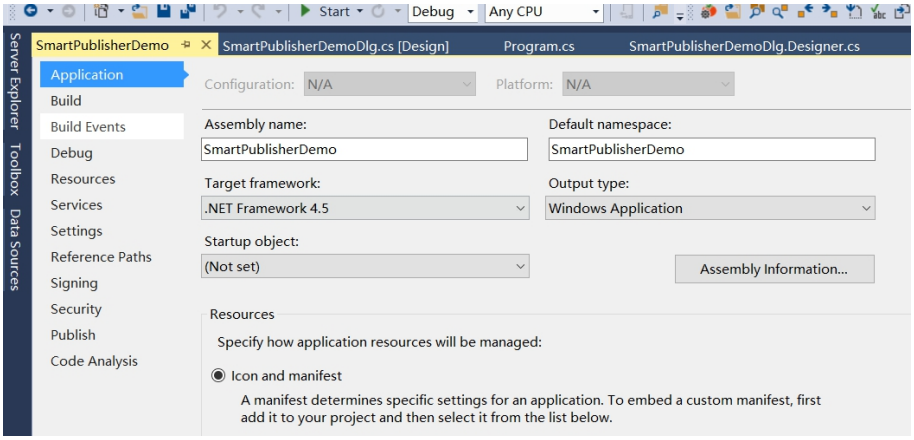
   (3) 64 位 debug 库：x64\debug

   (4) 64 位 release 库：x64\release

2. 相关 cs 头文件，加入需要集成的工程，对应 SmartPublisherDemo\SmartPublisherSDK 目录下的头文件；

3. 在需要集成的工程，右键->Properties->Application->Assembly name，**大牛直播 SDK 按照 APP 名称授权，未授权版本，此处请改成“SmartPublisherDemo”**，如需授权，可直接联系商

务：



4. 正式授权版，需要在 Init()接口调用之前添加设置 license 的代码（相关 Key 和 CID 请根据正式授权版邮件说明填写）。

1.4 功能详解

考虑到 Windows 平台推送端 SDK 功能相对复杂，以问答式：

1 视频采集设置

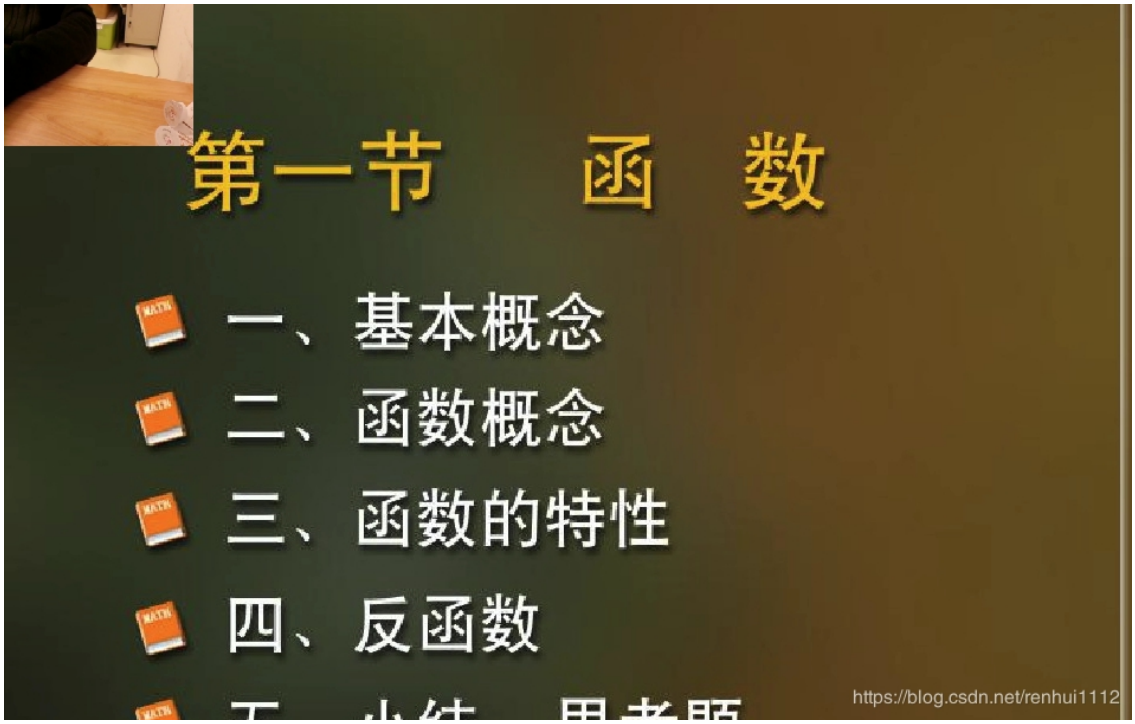
说明：

1. 屏幕和摄像头相互切换：用于在线教育或者无纸化等场景，推送或录像过程中，随时切换屏幕或摄像头数据（切换数据源），如需实时切换，点击页面“切换到摄像头”按钮即可；

2. 设置遮盖层，用于设定一个长方形或正方形区域（可自指定区域大小），遮盖不想给用户展示的部分；

3. 水印：添加 PNG 水印，支持推送或录像过程中，随时添加、取消水印；

4. 摄像头叠加到屏幕：意在用于同屏过程中，主讲人摄像头悬浮于屏幕之上（可指定叠加坐标），实现双画面展示，推送或录像过程中，可以随时取消摄像头叠加；



5. 屏幕叠加到摄像头：同 4，效果展示，实际根据需求实现；

6. 采集桌面：可以通过点击“选择屏幕区域”获取采集区域，并可在采集过程中，随时切换区域位置，如不设定，默认全屏采集；

7. 使用 DXGI 采集屏幕，采集时停用 Aero；

8. 采集窗口：可设定需要采集的窗口，窗口放大或缩小，推送端会自适应码率和分辨率；

9. 采集帧率(帧/秒)：默认屏幕采集 12 帧，可根据实际场景需求设定到期望帧率；

10. 缩放屏幕大小缩放比：用于高清或超高清屏，通过设定一定的比例因子，缩放屏幕采集分辨率；

11. 采集摄像头：可选择需要采集的摄像头、采集分辨率、帧率、是否需要水平或者垂直反转、是否需要旋转；

追加提问：

问题[确认数据源]：采集桌面还是摄像头？如果桌面，全屏还是部分区域？

回答：

如果是摄像头：可以选择摄像头列表，然后分辨率、帧率。

如果是屏幕：默认帧率是 12 帧，可以根据实际场景调整，选取屏幕区域，可以实时拉取选择需要采集或录像区域；

如果是叠加模式：可选择摄像头叠加到屏幕，还是屏幕叠加到摄像头；

更高需求的用户，可以设置水印或应用层遮盖。

问题：如果是摄像头，采集到的摄像头角度不对怎么办？

回答：我们支持摄像头镜像和翻转设置，摄像头可通过 SDK 接口轻松实现水平/垂直翻转、镜像效果。

2 视频码率控制

如何选择适合我的码率？

回答：如果不是有音视频背景的开发人员，可点击“获取视频码率默认值”，参考我们默认的码率推荐，如果觉得推荐码率过高或不够，可根据实际情况酌情调整。

265 编码还是 H.264 编码？

回答：Windows 平台支持 H.265 特定机型硬编码，如果推 RTMP 流，需要服务器支持 RTMP H.265 扩展，播放器 SDK，也需要同步支持 RTMP H.265 扩展播放。

如果是轻量级 RTSP 服务 SDK 对接的话，只需要播放器支持 RTSP H.265 即可。

如果推摄像头数据，建议采用可变码率+H.265 编码。

如何设置码率参数更合理？

回答：

关键帧间隔：一般来说，设置到帧率的 2-4 倍，比如帧率 20，关键帧间隔可以设置到 40-80；

平均码率：可以点击“获取视频码率默认值”，最大码率是平均码率的 2 倍；

视频质量：如果使用可变码率，建议采用大牛直播 SDK 默认推荐视频质量值；

编码速度：如高分辨率，建议 1-3，值越小，编码速度越快；

H.264 Profile：默认 baseline profile，可根据需要，酌情设置 High profile；

NOTE：点击“推送”或“录像”或启动内置 RTSP 服务 SDK 之前，请务必设置视频码率，如不想手动设置，请点击“获取视频码率默认值”！！！

3 音频采集设置

问答式：采集音频吗？如果采集，采集麦克风还是扬声器的，亦或混音？

回答：

如果想采集电脑输出的音频（比如音乐之类），可以选择“采集扬声器”；

如果想采集麦克风音频，可以选择“采集麦克风”，并选择相关设备；

如果两个都想采集，可以两个都选择，混音输出。

4 实时音量调节

问答式：采集过程中可以改变麦克风或扬声器采集音量吗？

回答：可以，如果二者都选中，处于混音模式，也可单独调整麦克风或扬声器音量。

5 音频编码

问题：是 AAC 还是 SPEEX？

回答：我们默认是 AAC 编码模式，如果需要码率更低，可以选择 SPEEX 编码模式，当然我们的 AAC 编码码率也不高，如果没有太高要求，考虑到通用性，建议使用 AAC。

6 音频处理

问题：我想过滤背景噪音怎么办？

回答：选中“噪音抑制”，“噪音抑制“请和“自动增益控制”组合使用，“端点检测(VAD)”可选设置。

问题：我想做一对一互动怎么办？

回答：选中“回音消除”，可以和“噪音抑制”、“自动增益控制”组合使用，具体可参看回音消除的 demo 工程：WIN-EchoCancellation-CSharp-Demo。

问题：我推送或者录像过程中，随时静音怎么办？

回答：推送过程中，随时选择或取消选择“静音”功能。

7 多路推送

问题：我想同时推送到多个 url 怎么办(比如一个内网服务器，一个外网服务器)？

回答：同时填写多个 url（最多 3 个），然后点推送即可。

8 截图(快照)

问题：我想推送或者录像过程中，截取当前图像怎么办？

回答：那就设置好截图路径，推送或录像过程中，随时点击“截图”。

9 录像

问题：我还想录像，怎么办？

回答：设置录像文件存放目录，文件前缀、单个文件大小，是否加日期、时间，随时录制即可，此外，我们的 SDK 还支持录像过程中，暂停录像，恢复录像。

10 实时预览

问题：我还想看看推出去视频，特别是合成后的效果，怎么办？

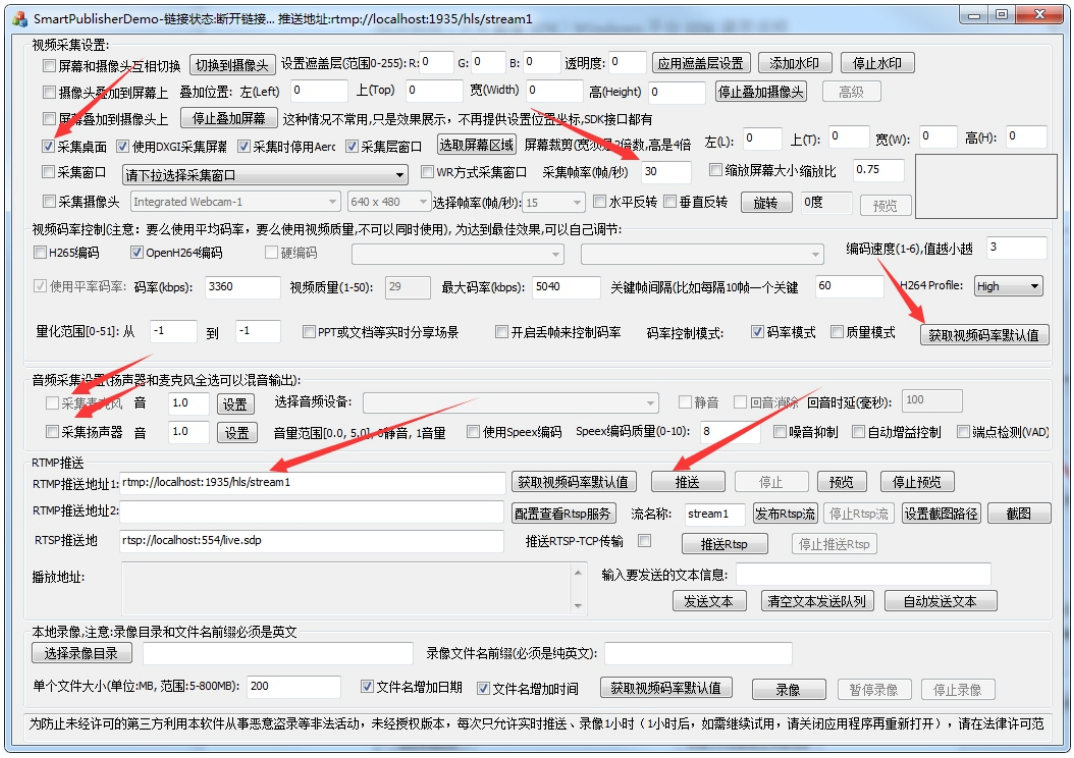
回答：点击页面的“预览”按钮，就可以看到。

11 如何快速测试

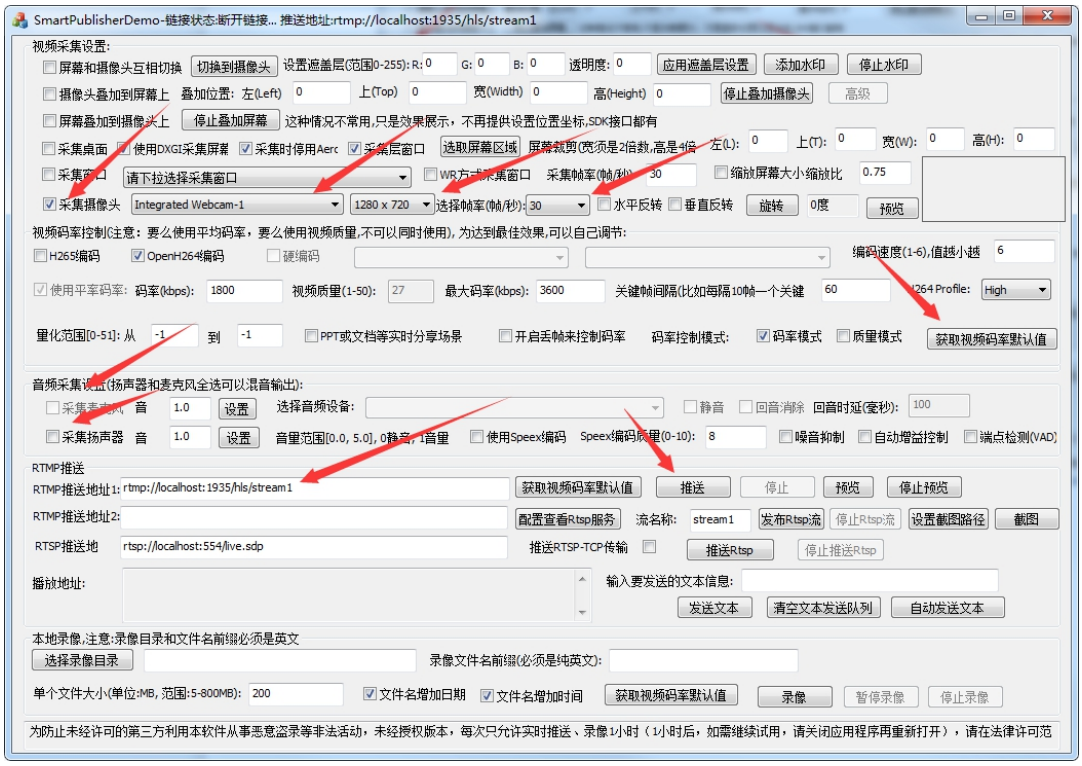
我是外行，我想快速测试推屏或推摄像头怎么办？

回答：

1.采集并推送屏幕：

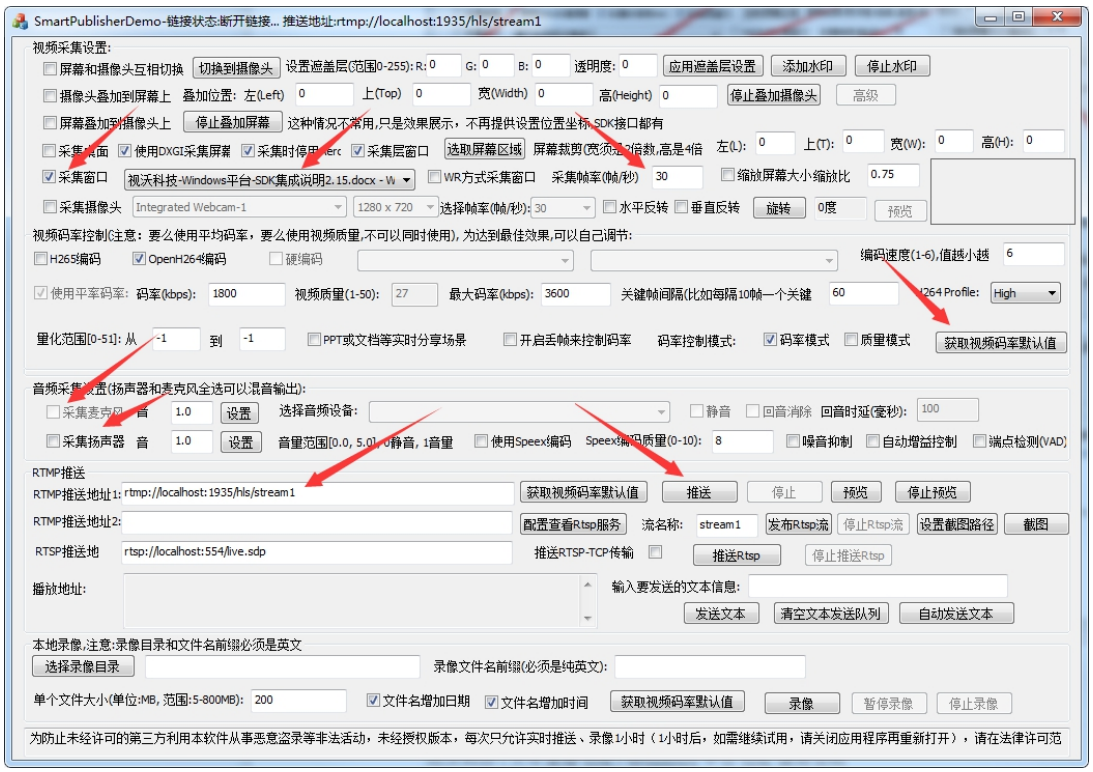


2.采集并推摄像头：



3.采集并推送窗口

不是所有的窗体都有权限采集到，考虑到兼容性，一般不建议使用这种模式。



1.5 接口调用时序(以 C#为例)

1.5.1 设置授权 license

C#的 SDK，请在 NTSmartPublisherSDK.NT_PB_Init 之前添加以下代码：

```
NTSmartPublisherSDK.NT_PB_SetSDKClientKey("xxxxxxxxxx", "xxxxxxxxxx", 0, IntPtr.Zero);
UInt32 isInited = NTSmartPublisherSDK.NT_PB_Init(0, IntPtr.Zero);
if (isInited != 0)
{
    return;
}
```

C++的 SDK，请在 player_api_.Init 之前添加下面的代码：

```
NT_SP_SetSDKClientKey("xxxxxxxxxx", "xxxxxxxxxx", 0, nullptr);
if ( NT_ERC_OK != player_api_.Init(0, NULL) )
{
     return FALSE;
}
```

1.5.2 设置日志存放路径

需要在 NT_PB_Init 之前添加下面的代码：

```
// 设置日志路径(请确保目录存在)
String log_path = "D:\\publisherlog";
NTSmartLog.NT_SL_SetPath(log_path);
```

如目录存在，并具备文件写入权限，关闭应用程序后，相关文件夹下会有 smart_sdk.log 生成。

1.5.3 初始化 SDK

NT_PB_Init：SDK 初始化，多实例推送，此接口仅需调用一次即可。

1.5.4 Open 生成推送实例

NT_PB_Open：通过调用 Open()接口，指定音视频数据源，并获取推送实例，对应 publisher_handle_；

```
/*定义 Video 源选项*/
public enum NT_PB_E_VIDEO_OPTION : uint
        {
        NT_PB_E_VIDEO_OPTION_NO_VIDEO = 0x0,
        NT_PB_E_VIDEO_OPTION_SCREEN = 0x1, // 采集屏幕
        NT_PB_E_VIDEO_OPTION_CAMERA = 0x2, // 摄像头采集
        NT_PB_E_VIDEO_OPTION_LAYER = 0x3,  // 视频合并，比如桌面叠加摄像头等
        NT_PB_E_VIDEO_OPTION_ENCODED_DATA = 0x4, // 已经编码的视频数据，目前支持 H264
        NT_PB_E_VIDEO_OPTION_WINDOW = 0x5, // 采集窗口
        }

/*定义 Audio 源选项*/
public enum NT_PB_E_AUDIO_OPTION : uint
        {
        NT_PB_E_AUDIO_OPTION_NO_AUDIO = 0x0,
        NT_PB_E_AUDIO_OPTION_CAPTURE_MIC = 0x1,          // 采集麦克风音频
        NT_PB_E_AUDIO_OPTION_CAPTURE_SPEAKER = 0x2,          // 采集扬声器
        NT_PB_E_AUDIO_OPTION_CAPTURE_MIC_SPEAKER_MIXER = 0x3,    // 麦克风扬声器混音
        NT_PB_E_AUDIO_OPTION_ENCODED_DATA = 0x4, // 编码后的音频数据，目前支持 AAC, speex 宽带(wideband mode)
        NT_PB_E_AUDIO_OPTION_EXTERNAL_PCM_DATA = 0x5, /*外部 PCM 数据*/
        NT_PB_E_AUDIO_OPTION_MIC_EXTERNAL_PCM_MIXER = 0x6, /* 麦克风和外部 PCM 数据混音
```

```
当前只支持一路外部音频和内置麦克风混音*/
        }
```

1.5.5 设置回调事件

(1) NT_PB_SetEventCallBack：设置事件回调，如果想监听事件的话，建议调用 Open 成功后，就调用这个接口
(2) NT_PB_SetVideoPacketTimestampCallBack：设置视频包时间戳回调
(3) NT_PB_SetPublisherStatusCallBack：设置推送状态回调

1.5.6 设置屏幕裁剪

(1) NT_PB_SetScreenClip：设置屏幕裁剪
(2) NT_PB_MoveScreenClipRegion：移动屏幕剪切区域，这个接口只能推送或者录像中调用

1.5.7 屏幕选取工具

(1) NT_PB_OpenScreenRegionChooseTool：打开一个屏幕选取工具的 toolHandle
(2) NT_PB_MoveScreenClipRegion：移动屏幕剪切区域，这个接口只能推送或者录像中调用
(3) NT_PB_AllocateImage：分配 Image，分配后，SDK 内部会初始化这个结构体，失败的话返回 NULL
(4) NT_PB_FreeImage：释放 Image，注意一定要调用这个接口释放内存，如果在你自己的模块中释放，Windows 会出问题的
(5) NT_PB_CloneImage：克隆一个 Image， 失败返回 NULL
(6) NT_PB_CopyImage：拷贝 Image，会先释放 dst 的资源，然后再拷贝
(7) NT_PB_SetImagePlane： 给图像一个面设置数据，如果这个面已经有数据，将会释放掉再设置
(8) NT_PB_LoadImage：加载 PNG 图片

1.5.8 设置屏幕|窗口采集参数

窗体采集受限于系统版本、权限等各个因素，不一定 100%能拿到数据，在可以直接可以使用屏幕采集的前提下，优先建议使用屏幕直接采集。

(1) NT_PB_EnableDXGIScreenCapturer：允许使用 DXGI 屏幕采集方式，这种方式需要 win8 及以上系统才支持
(2) NT_PB_DisableAeroScreenCapturer：采集屏幕时停用 Aero，这个只对 win7 有影响，win8 及以上系统，微软已经抛弃了 Aero Glass 效果
(3) NT_PB_EnableScreenCaptureLayeredWindow：在使用 GDI 方式采集屏幕时，如果需要采集 WS_EX_LAYERED 属性窗口,设置成 1，is_enable: 1 表示采集 WS_EX_LAYERED 属性窗口, 0 表示不采集 WS_EX_LAYERED 属性窗口. 默认系统是 0,不采集，注意采集 WS_EX_LAYERED 属性窗口,鼠标会闪烁。
(4) NT_PB_IsWRCaptureWindowSupported：检查是否支持 WR 方式采集窗口，这个需要 win10 较高版本才支持**[采集窗口]**
(5) NT_PB_SetCaptureWindowWay：设置捕获窗口方式，way: 1 使用 GDI DC 方式捕获, 2 使用 WR 方

式捕获，默认是 GDI DC 方式**[采集窗口]**

(6) NT_PB_CheckCapturerWindow：判断顶层窗口能否能被捕获，如果不能被捕获的话返回 NT_ERC_FAILED**[采集窗口]**

(7) NT_PB_CheckCapturerWindowV2：这个接口主要判断顶层窗口能否能被捕获，V2 版本，capture_way：这个请参考 SetCaptureWindowWay 说明，**[采集窗口]**

(8) NT_PB_SetCaptureWindow：设置要捕获的窗口的句柄**[采集窗口]**

1.5.9 设置摄像头采集参数

(1) NT_PB_StartGetVideoCaptureDeviceImage：获取句柄，且保存句柄

(2) NT_PB_FlipVerticalVideoCaptureDeviceImage：上下反转设备图像

(3) NT_PB_FlipHorizontalVideoCaptureDeviceImage：水平反转设备图像

(4) NT_PB_RotateVideoCaptureDeviceImage：旋转设备图像，顺时针旋转

(5) NT_PB_GetVideoCaptureDeviceNumber：获取摄像头数量

(6) NT_PB_GetVideoCaptureDeviceInfo：返回摄像头设备信息

(7) NT_PB_GetVideoCaptureDeviceCapabilityNumber：返回摄像头能力数

(8) NT_PB_GetVideoCaptureDeviceCapability:返回摄像头能力

(9) NT_PB_DisableVideoCaptureResolutionSetting:

(10) 在多个实例推送多路时，对于一个摄像头来说,所有实例只能共享摄像头，那么只有一个实例可以改变摄像头分辨率，其他实例使用这个缩放后的图像；

(11) 在使用多实例时，调用这个接口禁止掉实例的分辨率设置能力.只留一个实例能改变分辨，如果不设置，行为未定义；

(12) 这个接口必须在 SetLayersConfig, AddLayerConfig 之前调用。

(13) NT_PB_StartVideoCaptureDevicePreview: 启动摄像头预览

(14) NT_PB_FlipVerticalCameraPreview：上下反转摄像头预览图像

(15) NT_PB_FlipHorizontalCameraPreview：水平反转摄像头预览图像

(16) NT_PB_RotateCameraPreview：旋转摄像头预览图像，顺时针旋转

(17) NT_PB_VideoCaptureDevicePreviewWindowSizeChanged：告诉 SDK 预览窗口大小改变

(18) NT_PB_StopVideoCaptureDevicePreview：停止摄像头预览

(19) NT_PB_GetVideoCaptureDeviceImage：调用这个接口可以获取摄像头图像

(20) NT_PB_StopGetVideoCaptureDeviceImage：停止获取摄像头图像

(21) NT_PB_SetVideoCaptureDeviceBaseParameter：设置摄像头信息

(22) NT_PB_FlipVerticalCamera 上下反转摄像头图像

(23) NT_PB_FlipHorizontalCamera：水平反转摄像头图像

(24) NT_PB_RotateCamera：旋转摄像头图像，顺时针旋转

1.5.10 视频合成图层类型

```
/*视频合成时，每一层的类型*/
public enum NT_PB_E_LAYER_TYPE : int
        {
        NT_PB_E_LAYER_TYPE_SCREEN = 1,                  // 屏幕层
        NT_PB_E_LAYER_TYPE_CAMERA = 2,                  // 摄像头层
```

```
        NT_PB_E_LAYER_TYPE_RGBA_RECTANGLE = 3,          // RGBA 矩形
        NT_PB_E_LAYER_TYPE_IMAGE = 4,                   // 图片层
        NT_PB_E_LAYER_TYPE_EXTERNAL_VIDEO_FRAME = 5,    // 外部视频数据层
        NT_PB_E_LAYER_TYPE_WINDOW = 6,                  // 窗口层
        }
```

1.5.11 视频编码接口

(1) NT_PB_SetVideoEncoder：设置编码类型

```
/*
 * 设置软硬编码类型，编码器, codec_id，编码器其他参数.
 * type: 0 为软编码, 1 为硬编码，默认是软编码.
 * encoder_id: 如果是软编码，并且用 h264，可以设置 0, 0 用默认编码器，也可以设置 1，设置 1 将使用 OpenH264 编码. 如果不是 h264，请设置成 0；如果是硬编码, 128 为 NVIDIA video encoder (NVENC), 填其他值接口返回错误.
 * param1: 如果是软编码,请设置 0；如果是硬编码且是 NVENC，这个参数用来设置 GPU index，设置-1 的话 SDK 自动选择 GPU.
 * codec_id: 设置 h264 或 h265 编码，默认是 h264，请参考 NT_MEDIA_CODEC_ID， h264 填 NT_MEDIA_CODEC_ID_H264, h265 填 NT_MEDIA_CODEC_ID_H265.
 * 注意: 软编码不支持 h265，硬编码根据实际硬件情况决定是否支持 h265.
 * 如果调用了这个接口，请不要再调用 SetVideoEncoderType 接口
 * 成功返回 NT_ERC_OK
 */
    [DllImport(@"SmartPublisherSDK.dll")]
public static extern UInt32 NT_PB_SetVideoEncoder(IntPtr handle, Int32 type, Int32 encoder_id, UInt32 codec_id, Int32 param1);
```

(2) NT_PB_SetVideoQuality：设置视频质量，范围[0-20]，默认是 10，值越小质量越好，但码率会越大

(3) NT_PB_SetVideoQualityV2：设置视频质量，范围[1-50]，值越小视频质量越好,但码率会越大. 请优先考虑默认值；

(4) NT_PB_SetFrameRate：设置帧率

(5) NT_PB_SetVideoMaxBitRate：设置最大视频码率，单位 kbps

(6) NT_PB_AddVideoEncoderBitrateGroupItem：

* 在一些特殊场景下，视频分辨率会改变，如果设置一个固定码率的的话,当视频分辨率变大的时候会变的模糊,变小的话又会浪费码率
* 所以提供可以设置一组码率的接口,满足不同分辨率切换的需求
* 规则：比如设置两组分辨率 640*360, 640*480，那么当分辨率小于等于 640*360 时都使用 640*360 的码率,
* 当分辨率大于 640*360 且小于等于 640*480 时，就使用 640*480 的码率,如果分辨率大于 640*480 那就使用 640*480 的分辨率
* 为了设置的更准确，建议多划分几组，让区间变小
* 调用这个接口每次设置一组,设置多组就调用多次
* item 对应 NT_PB_VideoEncoderBitrateGroupItem

(7) NT_PB_ClearVideoEncoderBitrateGroup：清除视频码率组

(8) NT_PB_SetVideoKeyFrameInterval：设置关键帧间隔，比如 1 表示所有帧都是关键帧，10 表示每 10 帧里面一个关键帧，25 表示每 25 帧一个关键帧

(9) NT_PB_SetVideoEncoderProfile：设置 H264 profile，1: H264 baseline(默认值). 2: H264 main. 3. H264 high

(10) NT_PB_SetVideoEncoderSpeed：设置 H264 编码速度，speed: 范围是 1 到 6, 值越小，速度越快，质量也越差

1.5.12 音频编码接口

(1) NT_PB_GetAuidoInputDeviceNumber：获取系统音频输入设备数

(2) NT_PB_GetAuidoInputDeviceName：获取音频输入设备名称

(3) NT_PB_SetPublisherAudioCodecType：设置推送音频编码类型，type: 1:使用 AAC 编码，2:使用 speex 编码，其他值返回错误

(4) NT_PB_SetPublisherSpeexEncoderQuality：设置推送 Speex 编码质量

(5) NT_PB_SetAuidoInputDeviceId：设置音频输入设备 ID

(6) NT_PB_IsCanCaptureSpeaker：检查是否能捕获扬声器音频

(7) NT_PB_SetCaptureSpeakerCompensateMute：设置采集扬声器时是否补偿静音帧 这个只在 auido_option 是 NT_PB_E_AUDIO_OPTION_CAPTURE_SPEAKER 有效

1.5.13 音频处理接口

(1) NT_PB_SetEchoCancellation：设置回音消除

(2) NT_PB_SetNoiseSuppression：设置音频噪音抑制

(3) NT_PB_SetAGC：设置音频自动增益控制

(4) NT_PB_SetVAD：设置端点检测(Voice Activity Detection (VAD))

1.5.14 图层合成等接口

(1) NT_PB_SetLayersConfig：设置视频合成层， 传入的是一个数组， 请正确填充每一层

(2) NT_PB_ClearLayersConfig：清除所有层配置，注意这个接口只能在推送或者录像之前调用，否则结果未定义

(3) NT_PB_AddLayerConfig： 增加层配置，注意这个接口只能在推送或者录像之前调用，否则结果未定义

(4) NT_PB_EnableLayer：动态禁止或者启用层

(5) NT_PB_UpdateLayerConfigV2：更新层相关配置，注意不是层的所有字段都可以更新，只是部分可以更新，并且有些层没有字段可以更新，传入的参数，SDK 只选择能更新的字段更新，不能更新的字段会被忽略

(6) NT_PB_UpdateLayerRegion：修改图层

(7) NT_PB_PostLayerImage：给 index 层投递 Image 数据，目前主要是用来把 rgb 和 yuv 视频数据传给相关层

(8) NT_PB_SetParam：万能接口，设置参数， 大多数问题， 这些接口都能解决

(9) NT_PB_GetParam：万能接口，得到参数， 大多数问题，这些接口都能解决

1.5.15 RTMP 推送-设置推送 RTMP Url

NT_PB_SetURL：设置推送的 RTMP URL，如需推送到多个 RTMP 服务器，可设置多次，目前最大支持同时设置 3 个 RTMP URL。

```
/*
 * 设置推送的 URL
 *
 * 支持同时推送到多个 RTMP 服务器上，最多可以同时支持推到三个服务器上
 * 为设置多个 URL，请调用多次
 *
 * 成功返回 NT_ERC_OK
 */
[DllImport(@"SmartPublisherSDK.dll")]
public static extern UInt32 NT_PB_SetURL(IntPtr handle, [MarshalAs(UnmanagedType.LPStr)]String url,
IntPtr pReserve);
```

1.5.16 RTMP 推送-启动推送 RTMP 流

NT_PB_StartPublisher：根据设置的 RTMP 推流 URL，开始推送。

```
if (NTBaseCodeDefine.NT_ERC_OK != NTSmartPublisherSDK.NT_PB_StartPublisher(publisher_handle_,
IntPtr.Zero))
{
    if (0 == publisher_handle_count_)
    {
        NTSmartPublisherSDK.NT_PB_Close(publisher_handle_);
        publisher_handle_ = IntPtr.Zero;
    }

    is_publishing_ = false;

    MessageBox.Show("调用推流接口失败");

    return;
}
```

1.5.17 RTMP 推送-停止推送 RTMP 流

NT_PB_StopPublisher：停止推送。注意，此接口和 NT_PB_StartPublisher 配套使用。

```
NTSmartPublisherSDK.NT_PB_StopPublisher(publisher_handle_);
```

1.5.18 RTSP 推送-设置传输方式(TCP/UDP)

NT_PB_SetPushRtspTransportProtocol：设置推送 rtsp 传输方式，一般服务器可同时支持 RTSP TCP 或 UDP 传输模式，部分服务器只支持 TCP 或 UDP 模式。其中，transport_protocol: 1 表示 UDP 传输 rtp 包; 2 表示 TCP 传输 rtp 包. 默认是 1, UDP 传输。

1.5.19 RTSP 推送-设置推送 RTSP Url

NT_PB_SetPushRtspURL：注意，RTSP 推送时，确保服务器推送 URL 可用。

1.5.20 RTSP 推送-启动推送 RTSP 流

NT_PB_StartPushRtsp

1.5.21 RTSP 推送-启动推送 RTSP 流

NT_PB_StopPushRtsp：注意，此接口和 NT_PB_StartPushRtsp 配套使用。

1.5.22 RTMP/RTSP 推送端录像

(1) NT_PB_SetRecorderDirectory：设置本地录像目录，必须是英文目录，否则会失败
(2) NT_PB_SetRecorderFileMaxSize：设置单个录像文件最大大小，当超过这个值的时候，将切割成第二个文件
(3) NT_PB_SetRecorderFileNameRuler：设置录像文件名生成规则
(4) NT_PB_StartRecorder：启动录像
(5) NT_PB_PauseRecorder:暂停录像，is_pause: 1 表示暂停, 0 表示恢复录像，输入其他值将调用失败
(6) NT_PB_StopRecorder：停止录像

1.5.23 实时静音

NT_PB_SetMute：设置推送实时静音。

1.5.24 设置输入音量

NT_PB_SetInputAudioVolume
设置输入音量，这个接口一般不建议调用，在一些特殊情况下可能会用，一般不建议放大音量
index：一般是 0 和 1，如果没有混音的只用 0，有混音的话, 0,1 分别设置音量
volume：音量，默认是 1.0，范围是[0.0, 5.0]，设置成 0 静音, 1 音量不变

1.5.25 实时快照

NT_PB_CaptureImage：推送或者录像过程中，实时快照。

```
String name = capture_image_path_ + "\\" + DateTime.Now.ToString("hh-mm-ss") + ".png";

byte[] buffer1 = Encoding.Default.GetBytes(name);
byte[] buffer2 = Encoding.Convert(Encoding.Default, Encoding.UTF8, buffer1, 0, buffer1.Length);

byte[] buffer3 = new byte[buffer2.Length + 1];
buffer3[buffer2.Length] = 0;

Array.Copy(buffer2, buffer3, buffer2.Length);

IntPtr file_name_ptr = Marshal.AllocHGlobal(buffer3.Length);
        Marshal.Copy(buffer3, 0, file_name_ptr, buffer3.Length);

capture_image_call_back_ = new NT_PB_SDKCaptureImageCallBack(SDKCaptureImageCallBack);

UInt32 ret = NTSmartPublisherSDK.NT_PB_CaptureImage(publisher_handle_, file_name_ptr, IntPtr.Zero,
capture_image_call_back_);

Marshal.FreeHGlobal(file_name_ptr);

if (NT.NTBaseCodeDefine.NT_ERC_OK == ret)
{
    // 发送截图请求成功
}
else if
((UInt32)NT.NTSmartPublisherDefine.NT_PB_E_ERROR_CODE.NT_ERC_PB_TOO_MANY_CAPTURE_IMAGE_
REQUESTS == ret)
{
    // 通知用户延时
    MessageBox.Show("Too many capture image requests!");
}
else
{
    // 其他失败
}
```

1.5.26 关闭推送实例

NT_PB_Close：调用这个接口之后 handle 失效，每个推送实例调一次。

```
NTSmartPublisherSDK.NT_PB_Close(publisher_handle_);
publisher_handle_ = IntPtr.Zero;
```

1.5.27 UnInit

NT_PB_UnInit：这个是 SDK 最后一个调用的接口，多实例推送模式下，也仅需调用一次即可。

2. Windows 平台 RTMP/RTSP 直播播放 SDK

高稳定、超低延迟（一秒内，行业内几无效果接近的播放端）、业内首屈一指的 RTMP/RTSP 直播播放器 SDK。

2.1 demo 说明

- 大牛直播 SDK 提供 C++/C#两套接口，对外提供 32/64 位 debug/release 库，C++和 C#接口一一对应，C#接口比 C++接口增加前缀 NT_PB_；
- WIN-PlayerSDK-CPP-Demo：播放端 SDK 对应的 C++接口的 demo；
- WIN-PlayerSDK-CSharp-Demo：播放端 SDK 对应的 C#接口的 demo；
- 播放端 SDK 支持 Win7 及以上系统；
- 本 demo 基于 VS2013 开发。

2.2 界面 UI 展示

2.3 集成说明

C++头文件：

- [类型定义]nt_type_define.h
- [Log 定义]smart_log.h
- [Log 定义]smart_log_define.h
- [base code 定义]nt_base_code_define.h
- [player 接口]smart_player_define.h
- [player 参数定义]smart_player_sdk.h

C#头文件：

- [base code 定义]nt_base_code_define.cs
- [player 接口]smart_player_define.cs
- [player 参数定义]smart_player_sdk.cs

相关 Lib：

- **SmartLog.dll**
- **SmartLog.lib**
- **SmartPlayerSDK.dll**
- **SmartPlayerSDK.lib**
- avcodec-56.dll
- avdevice-56.dll
- avfilter-5.dll
- avformat-56.dll
- avutil-54.dll
- postproc-53.dll
- swresample-1.dll
- swscale-3.dll

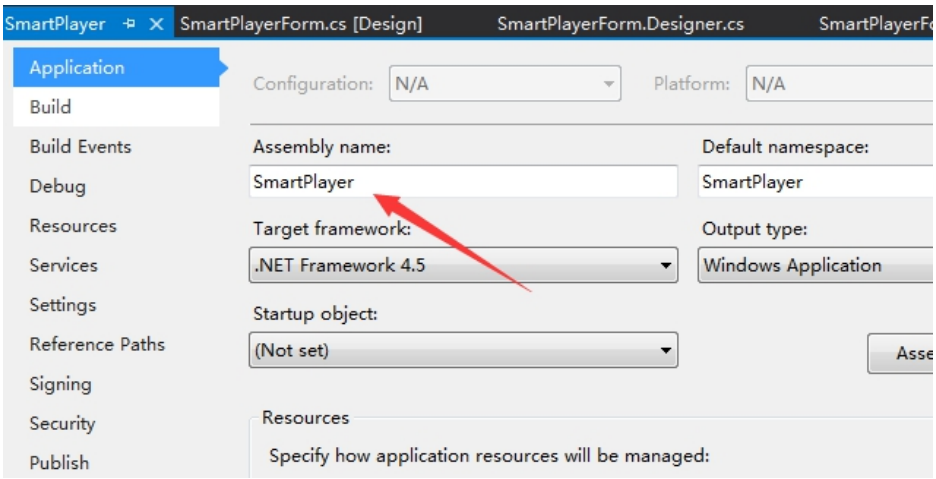
集成步骤：

3. 把 lib 目录下 debug/release 库拷贝到需要集成的工程对应的 debug 或 release 目录下（确保 32 位/64 位库 debug/release 目录一一对应）；

   lib 目录如下：

   (1) 32 位 debug 库：debug
   (2) 32 位 release 库：release
   (3) 64 位 debug 库：x64\debug
   (4) 64 位 release 库：x64\release

2. 相关 cs 头文件，加入需要集成的工程；
3. 在需要集成的工程，右键->Properties->Application->Assembly name，**大牛直播 SDK 按照 APP 名称授权，未授权版本，此处请改成“SmartPlayer”**，如需授权，可直接联系商务；



4. 正式授权版，需要在 Init()接口调用之前添加设置 license 的代码（相关 Key 和 CID 请根据正式授权版邮件说明填写）。

2.4 接口调用时序(以 C#为例)

2.4.1 设置授权 license

C#的 SDK，请在在 NT.NTSmartPlayerSDK.NT_SP_Init 之前添加下面的代码：

```
NT.NTSmartPlayerSDK.NT_SP_SetSDKClientKey("xxxxxxxxxx", "xxxxxxxxxx", 0, IntPtr.Zero);

UInt32 isInited = NT.NTSmartPlayerSDK.NT_SP_Init(0, IntPtr.Zero);
if (isInited != 0)
{
    MessageBox.Show("调用 NT_SP_Init 失败..");
    return;
}
```

C++的 SDK，请在 player_api_.Init 之前添加下面的代码：

```
NT_SP_SetSDKClientKey(NT_SP_SetSDKClientKey("xxxxxxxxxx", "xxxxxxxxxx", 0, nullptr);

if ( NT_ERC_OK != player_api_.Init(0, NULL) )
{
    return FALSE;
}
```

2.4.2 设置日志存放路径

需要在 player_api_.Init 之前添加下面的代码：

```
// 设置日志路径(请确保目录存在)
String log_path = "D:\\playerlog";
NTSmartLog.NT_SL_SetPath(log_path);
```

如目录存在，并具备文件写入权限，关闭应用程序后，相关文件夹下会有 smart_sdk.log 生成。

2.4.3 初始化 SDK

NT_SP_Init：SDK 初始化，多实例播放，此接口仅需调用一次即可。

2.4.4 特定机型硬解码检测

如系统用于特定机型环境下，特别是多路播放场景，需用到硬解码的话，可以用以下两组接口检测系统是否支持硬解。
注：在软解性能满足系统需求的前提下，一般建议优先使用软解。

```
/*
 * 检查是否支持 H264 硬解码
 * 如果支持的话返回 NT_ERC_OK
 */
[DllImport(@"SmartPlayerSDK.dll")]
public static extern UInt32 NT_SP_IsSupportH264HardwareDecoder();

/*
 * 检查是否支持 H265 硬解码
 * 如果支持的话返回 NT_ERC_OK
 */
[DllImport(@"SmartPlayerSDK.dll")]
public static extern UInt32 NT_SP_IsSupportH265HardwareDecoder();
```

如需使用硬解码，调用如下接口即可：

```
NTSmartPlayerSDK.NT_SP_SetH264HardwareDecoder(player_handle_,
is_support_h264_hardware_decoder_ ? 1 : 0, 0);
NTSmartPlayerSDK.NT_SP_SetH265HardwareDecoder(player_handle_,
is_support_h265_hardware_decoder_ ? 1 : 0, 0);
```

2.4.5 Open 生成播放实例

NT_SP_Open：每调用一次 Open 接口，对应一个播放实例，如需播放多实例，对应多个 player handler。

```
if (player_handle_ == IntPtr.Zero)
{
    player_handle_ = new IntPtr();

    UInt32 ret_open = NTSmartPlayerSDK.NT_SP_Open(out player_handle_, IntPtr.Zero, 0, IntPtr.Zero);

    if (ret_open != 0)
    {
        player_handle_ = IntPtr.Zero;
        MessageBox.Show("调用 NT_SP_Open 失败..");
        return;
```

```
    }
}
```

2.4.6 设置回调事件

(1) NT_SP_SetEventCallBack：用于回调网络链接状态、buffer 状态(开始、buffer 比例、结束)、实时带宽等，对应 EventID 如下：

```
/*事件 ID*/
public enum NT_SP_E_EVENT_ID : uint
{
        NT_SP_E_EVENT_ID_BASE = NTBaseCodeDefine.NT_EVENT_ID_SMART_PLAYER_SDK,

        NT_SP_E_EVENT_ID_CONNECTING              = NT_SP_E_EVENT_ID_BASE | 0x2, /*连接中*/
        NT_SP_E_EVENT_ID_CONNECTION_FAILED = NT_SP_E_EVENT_ID_BASE | 0x3, /*连接失败*/
        NT_SP_E_EVENT_ID_CONNECTED          = NT_SP_E_EVENT_ID_BASE | 0x4, /*已连接*/
        NT_SP_E_EVENT_ID_DISCONNECTED       = NT_SP_E_EVENT_ID_BASE | 0x5, /*断开连接*/
        NT_SP_E_EVENT_ID_NO_MEDIADATA_RECEIVED = NT_SP_E_EVENT_ID_BASE | 0x8,  /*收不到 RTMP 数据*/
        NT_SP_E_EVENT_ID_RTSP_STATUS_CODE    = NT_SP_E_EVENT_ID_BASE | 0xB,  /*rtsp status code 上报，目前只上报 401, param1 表示 status code*/

        /* 接下来请从 0x81 开始*/
        NT_SP_E_EVENT_ID_START_BUFFERING = NT_SP_E_EVENT_ID_BASE | 0x81, /*开始缓冲*/
        NT_SP_E_EVENT_ID_BUFFERING       = NT_SP_E_EVENT_ID_BASE | 0x82, /*缓冲中, param1 表示百分比进度*/
        NT_SP_E_EVENT_ID_STOP_BUFFERING  = NT_SP_E_EVENT_ID_BASE | 0x83, /*停止缓冲*/

        NT_SP_E_EVENT_ID_DOWNLOAD_SPEED  = NT_SP_E_EVENT_ID_BASE | 0x91, /*下载速度，param1 表示下载速度，单位是(Byte/s)*/

        NT_SP_E_EVENT_ID_PLAYBACK_REACH_EOS = NT_SP_E_EVENT_ID_BASE | 0xa1,      /*播放结束，直播流没有这个事件，点播流才有*/
        NT_SP_E_EVENT_ID_RECORDER_REACH_EOS = NT_SP_E_EVENT_ID_BASE | 0xa2,      /*录像结束，直播流没有这个事件，点播流才有*/
        NT_SP_E_EVENT_ID_PULLSTREAM_REACH_EOS = NT_SP_E_EVENT_ID_BASE | 0xa3,   /*拉流结束，直播流没有这个事件，点播流才有*/

        NT_SP_E_EVENT_ID_DURATION = NT_SP_E_EVENT_ID_BASE | 0xa8, /*视频时长，如果是直播，则不上报，如果是点播的话，若能从视频源获取视频时长的话，则上报， param1 表示视频时长，单位是毫秒(ms)*/
}
```

(2) NT_SP_SetVideoSizeCallBack：设置视频分辨率回调，如流数据携带视频数据，SDK 会回上来视频宽高信息：

```
//video resolution callback
video_size_call_back_ = new SP_SDKVideoSizeCallBack(SP_SDKVideoSizeHandle);
NTSmartPlayerSDK.NT_SP_SetVideoSizeCallBack(player_handle_, IntPtr.Zero, video_size_call_back_);
```

注意：视频宽高回上来或绘制窗口发生变化时，记得调用 NT_SP_OnWindowSize()更新，如不调用可能会引起视频模糊。

```
private void PlaybackWindowResized(Int32 width,Int32 height)
{
    width_=width;
    height_=height;

    int left=playWnd.Left;
    int top=playWnd.Top;

    textBox_resolution.Text=width+"*"+height;

    if(player_handle_==IntPtr.Zero)
    {
        return;
    }

    NTSmartPlayerSDK.NT_SP_OnWindowSize(player_handle_,playWnd.Width,playWnd.Height);
}
```

(3) NT_SP_SetVideoFrameCallBack：设置 YUV/RGB32 数据回调，可用于对接第三方视频分析，或自行绘制等，如系统不支持 D3D 绘制，可设置回调数据，上层 GDI 模式绘制：

```
/*定义视频帧图像格式*/
public enum NT_SP_E_VIDEO_FRAME_FORMAT : uint
{
    NT_SP_E_VIDEO_FRAME_FORMAT_RGB32 = 1, // 32 位的rgb 格式, r, g, b 各占8，另外一个字节保留，内存字节格式为: bb gg rr xx，主要是和 windows 位图匹配，在小端模式下，按 DWORD 类型操作，最高位是 xx，依次是 rr, gg, bb
    NT_SP_E_VIDEO_FRAME_FORMAT_ARGB = 2, // 32 位的 argb 格式，内存字节格式是: bb gg rr aa 这种类型，和 windows 位图匹配
    NT_SP_E_VIDEO_FRAME_FROMAT_I420 = 3, // YUV420 格式，三个分量保存在三个面上
}
```

(4) NT_SP_SetVideoFrameCallBackV2：设置 YUV/RGB32 数据回调，与 NT_SP_SetVideoFrameCallBack 接口的不同在于，吐出来的视频数据，可以指定宽高；

(5) NT_SP_SetRenderVideoFrameTimestampCallBack：设置绘制视频帧时，视频帧时间戳回调，一般播放器无时间戳回调需求的话，无需设置：

```
//video timestamp callback
video_frame_ts_callback_ = new
SP_SDKRenderVideoFrameTimestampCallBack(SP_SDKRenderVideoFrameTimestampCallBack);
NTSmartPlayerSDK.NT_SP_SetRenderVideoFrameTimestampCallBack(player_handle_, IntPtr.Zero,
video_frame_ts_callback_);
```

(6) NT_SP_SetAudioPCMFrameCallBack：设置音频 PCM 帧回调，吐 PCM 数据出来，目前每帧大小是 10ms，一般播放器无使用需求的话，无需设置；

(7) NT_SP_SetUserDataCallBack：设置用户数据回调，此接口需要和推送端 SDK 配套使用，用于返回推送端设定的实时用户数据（如时间戳、经纬度等各种扩展指令或信息），如只是单纯使用播放 SDK，无需设置；

(8) NT_SP_SetSEIDataCallBack：设置视频 SEI 数据回调，如只是单纯使用播放 SDK，不需要额外处理扩展 SEI 数据的话，无需设置。

2.4.7 D3DRender 检测

目前，几乎很少存在不支持 D3D 绘制的情况，考虑到系统通用性，我们在播放之前，先做检测，具体调用接口如下：

```
/*
 * handle: 播放句柄
 * hwnd: 这个要传入真正用来绘制的窗口句柄
 * is_support: 如果支持的话 *is_support 为1， 不支持的话为0
 * 接口调用成功返回 NT_ERC_OK
 */
[DllImport(@"SmartPlayerSDK.dll")]
public static extern UInt32 NT_SP_IsSupportD3DRender(IntPtr handle, IntPtr hwnd, ref Int32 is_support);
```

对于不支持 D3D 绘制的情况下，设置回调 YUV 数据，上层直接用 GDI 模式绘制，注意：GDI 绘制效率偏低。

```
Int32 in_support_d3d_render = 0;

if (NT.NTBaseCodeDefine.NT_ERC_OK == NTSmartPlayerSDK.NT_SP_IsSupportD3DRender(player_handle_,
playWnd.Handle, ref in_support_d3d_render))
{
    if (1 == in_support_d3d_render)
    {
        is_support_d3d_render = true;
    }
}

if (is_support_d3d_render)
```

```
{
    is_gdi_render_ = false;

    // 支持 d3d 绘制的话，就用 D3D 绘制
    NTSmartPlayerSDK.NT_SP_SetRenderWindow(player_handle_, playWnd.Handle);

    if (btn_check_render_scale_mode.Checked)
    {
        NTSmartPlayerSDK.NT_SP_SetRenderScaleMode(player_handle_, 1);
    }
    else
    {
        NTSmartPlayerSDK.NT_SP_SetRenderScaleMode(player_handle_, 0);
    }
}
else
{
    is_gdi_render_ = true;
    playWnd.Visible = false;

    // 不支持 D3D 就让播放器吐出数据来，用 GDI 绘制
    //video frame callback (YUV/RGB)
    //format 请参见 NT_SP_E_VIDEO_FRAME_FORMAT，如需回调 YUV，请设置为
NT_SP_E_VIDEO_FRAME_FROMAT_I420
    video_frame_call_back_ = new SP_SDKVideoFrameCallBack(SetVideoFrameCallBack);
NTSmartPlayerSDK.NT_SP_SetVideoFrameCallBack(player_handle_,
(Int32)NT.NTSmartPlayerDefine.NT_SP_E_VIDEO_FRAME_FORMAT.NT_SP_E_VIDEO_FRAME_FORMAT_RG
B32, IntPtr.Zero, video_frame_call_back_);
}
```

2.4.8 设置播放 URL

NT_SP_SetURL：支持 rtsp/rtmp/本地 FLV 文件(全路径)。

2.4.9 设置回调 PCM 数据

NT_SP_SetIsOutputAudioDevice：设置是否播放出声音，这个和静音接口是有区别的，这个接口的主要目的是为了用户设置了外部 PCM 回调接口后，又不想让 SDK 播放出声音时使用。

```
/*
 * 设置是否播放出声音，这个和静音接口是有区别的
 * 这个接口的主要目的是为了用户设置了外部 PCM 回调接口后，又不想让 SDK 播放出声音时使用
 * is_output_auido_device: 1: 表示允许输出到音频设备，默认是 1， 0：表示不允许输出. 其他值接口
```

```
返回失败
 * 成功返回 NT_ERC_OK
 */
[DllImport(@"SmartPlayerSDK.dll")]
public static extern UInt32 NT_SP_SetIsOutputAudioDevice(IntPtr handle, Int32 is_output_auido_device);
```

2.4.10 设置回调 YUV/RGB 数据

为了便于开发者更高级别的客制化对接，如做视频分析、自行绘制等，当然如上面 NT_SP_SetVideoFrameCallBack 提到的，如系统不支持 D3D 绘制，也可设置回调数据，上层 GDI 模式绘制，SDK 主要支持 RGB32、ARGB、I420 三种数据类型回调：

```
/*定义视频帧图像格式*/
public enum NT_SP_E_VIDEO_FRAME_FORMAT : uint
{
    NT_SP_E_VIDEO_FRAME_FORMAT_RGB32 = 1, // 32 位的rgb 格式, r, g, b 各占8, 另外一个字节保留, 内存字节格式为: bb gg rr xx, 主要是和 windows 位图匹配, 在小端模式下，按 DWORD 类型操作，最高位是 xx, 依次是 rr, gg, bb
    NT_SP_E_VIDEO_FRAME_FORMAT_ARGB = 2, // 32 位的 argb 格式，内存字节格式是: bb gg rr aa 这种类型，和 windows 位图匹配
    NT_SP_E_VIDEO_FRAME_FROMAT_I420 = 3, // YUV420 格式, 三个分量保存在三个面上
}
```

如果需要指定视频宽高信息回调，需用到 NT_SP_SetVideoFrameCallBackV2 这个回调设定：同样是设置 YUV/RGB32 数据回调，它与 NT_SP_SetVideoFrameCallBack 接口的不同之处在于，吐出来的视频数据，可以指定宽高；

2.4.11 RTMP/RTSP 播放参数设置

具体可参照 Demo 源码里面 InitCommonSDKParam()：

2.4.11.1 播放前可选设置接口

(1) NT_SP_SetBuffer：设置视频播放缓冲 buffer 大小，单位：毫秒；

(2) NT_SP_SetRTSPTcpMode：设置 RTSP TCP 模式, 1 为 TCP, 0 为 UDP, 此接口仅 RTSP 有效；

(3) NT_SP_SetRtspTimeout：设置 RTSP 超时时间, timeout 单位为秒，必须大于 0，一般建议 8-12 秒；

(4) NT_SP_SetRtspAutoSwitchTcpUdp：对于 RTSP 来说，有些可能支持 rtp over udp 方式，有些可能支持使用 rtp over tcp 方式. 为了方便使用，有些场景下可以开启自动尝试切换开关, 打开后如果 udp 无法播放，sdk 会自动尝试 tcp，如果 tcp 方式播放不了,sdk 会自动尝试 udp，is_auto_switch_tcp_udp： 如果设置 1 的话, sdk 将在 tcp 和 udp 之间尝试切换播放，如果设置为 0，则不尝试切换；

(5) NT_SP_SetFastStartup：设置秒开, 1 为秒开, 0 为不秒开，此接口用于如 RTMP 服务器缓存 GOP 时，

酌情使用；

(6) NT_SP_SetLowLatencyMode：设置低延时播放模式，默认是正常播放模式，mode: 1 为低延时模式， 0 为正常模式，低延迟模式下，可能会导致音视频不同步，或视频帧不均匀；

(7) NT_SP_SetReportDownloadSpeed：设置下载速度上报，默认不上报下载速度；

```
/*
 * 设置下载速度上报，默认不上报下载速度
 * is_report: 上报开关, 1: 表上报. 0: 表示不上报. 其他值无效.
 * report_interval： 上报时间间隔（上报频率），单位是秒，最小值是 1 秒 1 次. 如果小于 1 且设置了上报，将调用失败
 * 注意：如果设置上报的话，请设置 SetEventCallBack，然后在回调函数里面处理这个事件.
 * 上报事件是：NT_SP_E_EVENT_ID_DOWNLOAD_SPEED
 * 这个接口必须在 StartXXX 之前调用
 * 成功返回 NT_ERC_OK
 */
[DllImport(@"SmartPlayerSDK.dll")]
public static extern UInt32 NT_SP_SetReportDownloadSpeed(IntPtr handle, Int32 is_report, Int32 report_interval);
```

(8) NT_SP_GetDownloadSpeed：主动获取下载速度，speed： 返回下载速度，单位是 Byte/s；

(9) NT_SP_SetParam：万能接口，设置参数， 大多数问题， 这些接口都能解决；

(10) NT_SP_GetParam：万能接口，得到参数， 大多数问题，这些接口都能解决；

2.4.11.2 播放前后可实时调用的接口

(1) NT_SP_SetMute：播放过程中，实时静音、取消静音，可播放之前调用，亦或播放过程中实时调用；

(2) NT_SP_SetAudioVolume：不同于实时静音接口，此接口可以更细粒度的控制音量，默认范围[0,100]，其中 0 是静音，100 是最大音量，默认是 100，这个接口实际上覆盖了 NT_SP_SetMute 的功能；

(3) NT_SP_SetOnlyDecodeVideoKeyFrame：多窗口播放场景下，部分窗口可能只需要播放关键帧，如有类似场景需求，可用此接口；

(4) NT_SP_SetRotation：设置视频 View 旋转，顺时针旋转，degress： 设置 0， 90， 180， 270 度有效，其他值无效，注意：除了 0 度，其他角度播放会耗费更多 CPU；

(5) NT_SP_SetFlipVertical：设置视频 View 上下反转(垂直反转)；

(6) NT_SP_SetFlipHorizontal：设置视频 View 水平反转；

(7) NT_SP_SetRenderScaleMode:设置视频画面的填充模式，如填充整个绘制窗口、等比例填充绘制窗口，如不设置，默认填充整个绘制窗口；

如果是 D3D 模式，是通过此接口，直接设置画面填充模式，极少数设备不支持 D3D 的话，是数据回上来 RGB 的数据，上层绘制的，这时可参看我们绘制的代码，也实现了填充模式设置支持。

2.4.12 开始播放

NT_SP_StartPlay

开始播放 RTMP 或 RTSP 流数据。

2.4.13 RTMP/RTSP 拉流端录像

(1) NT_SP_SetRecorderDirectory：设置录像目录
(2) NT_SP_SetRecorderFileMaxSize：设置单个文件最大大小
(3) NT_SP_SetRecorderFileNameRuler：设置录像文件名生成规则
(4) NT_SP_SetRecorderCallBack：设置录像回调接口
(5) NT_SP_SetRecorderAudioTranscodeAAC：设置录像时音频转 AAC 编码的开关, aac 比较通用，sdk 增加其他音频编码(比如 speex, pcmu, pcma 等)转 aac 的功能
(6) NT_SP_SetRecorderVideo：设置是否录视频，默认的话，如果视频源有视频就录，没有就没得录，但有些场景下可能不想录制视频，只想录音频，所以增加个开关
(7) NT_SP_SetRecorderAudio：设置是否录音频，默认的话，如果视频源有音频就录，没有就没得录，但有些场景下可能不想录制音频，只想录视频，所以增加个开关
(8) NT_SP_StartRecorder：启动录像
(9) NT_SP_StopRecorder：停止录像

```
NTSmartPlayerSDK.NT_SP_SetRecorderFileMaxSize(player_handle_, max_file_size_);
NT_SP_RecorderFileNameRuler rec_name_ruler = new NT_SP_RecorderFileNameRuler();

rec_name_ruler.type_ = 0;
rec_name_ruler.file_name_prefix_ = rec_name_file_prefix_;
rec_name_ruler.append_date_ = is_append_date_ ? 1 : 0;
rec_name_ruler.append_time_ = is_append_time_ ? 1 : 0;

NTSmartPlayerSDK.NT_SP_SetRecorderFileNameRuler(player_handle_, ref rec_name_ruler);
record_call_back_ = new SP_SDKRecorderCallBack(SDKRecorderCallBack);
NTSmartPlayerSDK.NT_SP_SetRecorderCallBack(player_handle_, IntPtr.Zero, record_call_back_);
NTSmartPlayerSDK.NT_SP_SetRecorderAudioTranscodeAAC(player_handle_, is_audio_transcode_aac_ ? 1 :
0);

if (NT.NTBaseCodeDefine.NT_ERC_OK != NTSmartPlayerSDK.NT_SP_StartRecorder(player_handle_))
{
    MessageBox.Show("录像失败!");
    return;
}
```

2.4.14 实时快照

NT_SP_CaptureImage
用于播放端实时截取当前播放图片，图片以 PNG 形式保存至本地。

```
String name = capture_image_path_ + "\\" +    DateTime.Now.ToString("hh-mm-ss") + ".png";

byte[] buffer1 = Encoding.Default.GetBytes(name);
byte[] buffer2 = Encoding.Convert(Encoding.Default, Encoding.UTF8, buffer1, 0, buffer1.Length);

byte[] buffer3 = new byte[buffer2.Length + 1];
buffer3[buffer2.Length] = 0;

Array.Copy(buffer2, buffer3, buffer2.Length);

IntPtr file_name_ptr = Marshal.AllocHGlobal(buffer3.Length);
Marshal.Copy(buffer3, 0, file_name_ptr, buffer3.Length);

capture_image_call_back_ = new SP_SDKCaptureImageCallBack(SDKCaptureImageCallBack);

UInt32 ret = NTSmartPlayerSDK.NT_SP_CaptureImage(player_handle_, file_name_ptr, IntPtr.Zero,
capture_image_call_back_);

Marshal.FreeHGlobal(file_name_ptr);

if (NT.NTBaseCodeDefine.NT_ERC_OK == ret)
{
    // 发送截图请求成功
}
else if
((UInt32)NT.NTSmartPlayerDefine.SP_E_ERROR_CODE.NT_ERC_SP_TOO_MANY_CAPTURE_IMAGE_REQUE
STS == ret)
{
    // 通知用户延时
    MessageBox.Show("Too many capture image requests!");
}
else
{
    // 其他失败
}
```

2.4.15 快速切换 URL

NT_SP_SwitchURL
快速切换 URL，用于不用析构整个 player 实例的前提下，实时切换播放的 URL。

2.4.16 用户数据回调

NT_SP_SetUserDataCallBack
设置用户数据回调，用于接收扩展 SEI 模块发送的用户数据信息，如不是配合我们扩展 SEI 发送 DK，此接口无需调用。

2.4.17 SEI 数据回调

NT_SP_SetSEIDataCallBack
设置视频 sei 数据回调，用于接收 SEI 数据回调，如流数据不存在 SEI 或不准备处理 SEI 数据，此接口无需调用。

2.4.18 停止播放

NT_SP_StopPlay
停止播放 RTMP 或 RTSP 流数据。

2.4.19 关闭播放实例

NT_SP_Close
调用 Close 接口后，player handler 置空。

```
if ( player_handle_ != IntPtr.Zero)
{
        NTSmartPlayerSDK.NT_SP_Close(player_handle_);
        player_handle_ = IntPtr.Zero;
}
```

2.4.20 Uninit

NT_SP_UnInit
UnInit() 是 SDK 最后一个调用的接口，多实例环境下，只需要调用一次即可。

3 Windows 平台内置轻量级 RTSP 服务 SDK

为满足内网无纸化/电子教室等内网超低延迟需求，避免让用户配置单独的服务器，大牛直播 SDK 在推送端发布了轻量级 RTSP 服务 SDK。

简单来说，之前推送端 SDK 支持的功能，内置轻量级 RTSP 服务 SDK 后，功能继续支持。

内置轻量级 RTSP 服务后，延迟更低，体验更好。

3.1 功能说明

- [基础功能]支持 Windows 平台 RTMP 直播 SDK 除推送 RTMP 外的所有常规功能；
- [音频格式]AAC；
- [视频格式]H.264、H.265；
- [协议类型]RTSP；
- [传输模式]支持单播和组播模式；
- [端口设置]支持 RTSP 端口设置；
- [鉴权设置]支持 RTSP 鉴权用户名、密码设置；
- [获取 session 连接数]支持获取当前 RTSP 服务会话连接数；
- [多服务支持]支持同时创建多个内置 RTSP 服务；
- [H.265 支持]Windows 内置 rtsp server 支持发布 H.265（特定机型硬编码）视频；
- [RTSP url 回调]支持设置后的 rtsp url 通过 event 回调到上层。

3.2 对应 Demo

- Windows 测试程序：SmartPublisherDemo.exe；
- Windows C++工程：WIN-PublisherSDK-CPP-Demo；
- Windows C#工程：WIN-PublisherSDK-CSharp-Demo。

3.3 接口详解

Windows 内置轻量级 RTSP 服务 SDK 接口详解		
调用描述	接口	接口描述
SmartRTSPServerSDK		

创建一个 rtsp server	NT_PB_OpenRtspServer	创建一个 rtsp server，返回 rtsp server 句柄
设置端口	NT_PB_SetRtspServerPort	设置 rtsp server 监听端口，在 StartRtspServer 之前必须要设置端口
设置鉴权用户名、密码	NT_PB_SetRtspServerUserNamePassword	设置 rtsp server 鉴权用户名和密码，这个可以不设置，只有需要鉴权的再设置
获取 rtsp server 当前会话数	NT_PB_GetRtspServerClientSessionNumbers	获取 rtsp server 当前的客户会话数，这个接口必须在 StartRtspServer 之后再调用
启动 rtsp server	NT_PB_StartRtspServer	启动 rtsp server
停止 rtsp server	NT_PB_StopRtspServer	停止 rtsp server
关闭 rtsp server	NT_PB_CloseRtspServer	关闭 rtsp server
SmartRTSPServerSDK 供 Publisher 调用的接口		
设置 rtsp 的流名称	NT_PB_SetRtspStreamName	设置 rtsp 的流名称
给要发布的 rtsp 流设置 rtsp server	NT_PB_AddRtspStreamServer	给要发布的 rtsp 流设置 rtsp server，一个流可以发布到多个 rtsp server 上，rtsp server 的创建启动请参考 OpenRtspServer 和 StartRtspServer 接口
清除设置的 rtsp server	NT_PB_ClearRtspStreamServer	清除设置的 rtsp server
启动 rtsp 流	NT_PB_StartRtspStream	启动 rtsp 流
停止 rtsp 流	NT_PB_StopRtspStream	停止 rtsp 流

4 Windows 平台 RTMP/RTSP 多路流媒体转 RTMP 推送 SDK

Windows 转发 SDK，简单来说，播放端 SDK 拉取 RTSP/RTMP 流，并回调编码后的音视频数据到上层，然后，调用我们的推送端 SDK，通过推送端 SDK 扩展数据接口，完成 RTMP 数据转发，整个过程由于不涉及解码、重编码，支持多路转发，超低延迟和低资源占用。

视沃科技（大牛直播 SDK）多路 RTMP/RTSP 转 RTMP 转发软件，系原有转发 SDK 基础上，官方推出的 Windows 平台定制版。在秉承低延迟、灵活稳定、低资源占用的前提下，客户无需关注开发细节，只需图形化配置转发等各类参数，实现产品快速上线目的。

如监控类摄像机、NVR 等，通过厂商说明或 Onvif 工具，获取拉流的 RTSP 地址，图形化配置，完成拉流转发等操作，轻松实现标准 RTMP 服务器（或 CDN）对接。

视频转发支持 H.264、H.265（需要 RTMP 服务器或 CDN 支持扩展 H.265），音频支持配置 PCMA/PCMU 转 AAC 后转发，并支持只转发/录制视频或音频，RTSP 拉流端支持鉴权和 TCP/UDP 模式设置和 TCP/UDP 模式自动切换，整个拉流、转发模块都有非常完善的自动重连机制。

此外，可以通过点击拉流地址或推流地址栏，实现推拉流地址，同步到左侧预览框，实现推拉流音视频数据预览。

运维方面，官方定制版转发系统支持 7*24 小时不间断运行，自带守护进程，转发程序被误关等各种操作后，会自动启动运行，此外，还支持开机自动启动转发或录像。

4.1 功能说明

Windows 多路流媒体转发 SDK	
功能	功能描述
标准功能	通过“**Windows 播放端 SDK**”拉流，回调 H.264/AAC/SPEEX/PCMA/PCMU 数据，调用“**Windows 推送端 SDK**”外置数据接口，实现转发
拉流音频转码	支持拉取的 RTMP/RTSP 的 PCMA/PCMU/SPEEX 音频格式**转 AAC** 后再转发到

	RTMP 服务器
多实例	支持同时转发多路 RTMP/RTSP 音视频流
本地预览	因支持“**Windows 播放端 SDK**”功能，支持转发过程中，随时本地预览
拉流音频调节	支持拉取的 RTMP/RTSP 流实时静音
切换转发源	支持转发过程中，随时切换拉流的数据源(源 URL 变化，推流 URL 不变)，播放端 SDK 无感知(还是同一个拉流 URL)低延迟播放切换后数据源
逻辑分离	支持播放、录像、转发逻辑完全分离，三者可随意组合或单独使用
H.265(HEVC)支持	业内为数不多**支持 RTSP/RTMP H.265 转 RTMP 推送**的 SDK(提供配套 RTMP 扩展 H.265 服务器)；

4.2 对应 Demo

- ✧ Windows 测试程序：SmartStreamRelayDemo.exe；
- ✧ Windows C++工程：WIN-RelaySDK-CPP-Demo；
- ✧ Windows C#工程：WIN-RelaySDK-CSharp-Demo。

4.3 接口详解

Windows 转发 SDK 接口详解		
调用描述	**接口**	**接口描述**
创建推流实例和播放实例	从播放端拉取 RTSP/RTMP 流，并回调到上层，调用推送端数据转发接口，实现转发逻辑	注意：推送端调用 NT_PB_Open 时： Audio_opt： NT_PB_E_AUDIO_OPTION_ENCODED_DATA video_opt: NT_PB_E_VIDEO_OPTION_ENCODED_DATA.
播放端开始拉流	NT_SP_StartPullStream	播放端开始拉流，用于音视频数据转发
播放端停止拉流	NT_SP_StopPullStream	播放端停止拉流，用于音视频数据转发
播放端视频回调	NT_SP_SetPullStreamVideoDataCallBack	回调视频数据
播放端音频回调	NT_SP_SetPullStreamAudioDataCallBack	回调音频数据
音频转码	NT_SP_SetPullStreamAudioTr	设置拉流时音频转 AAC 编码的开关

	anscodeAAC	(PCMA/PCMU/SPEEX 转 AAC)
视频转发	NT_PB_PostVideoEncodedDataV2	设置编码后视频数据(H.264)
音频转发	NT_PB_PostAudioEncodedData	设置音频数据(AAC/PCMA/PCMU/SPEEX)