视沃科技

大牛直播 SDK

Unity3D 调用说明

官网：http://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

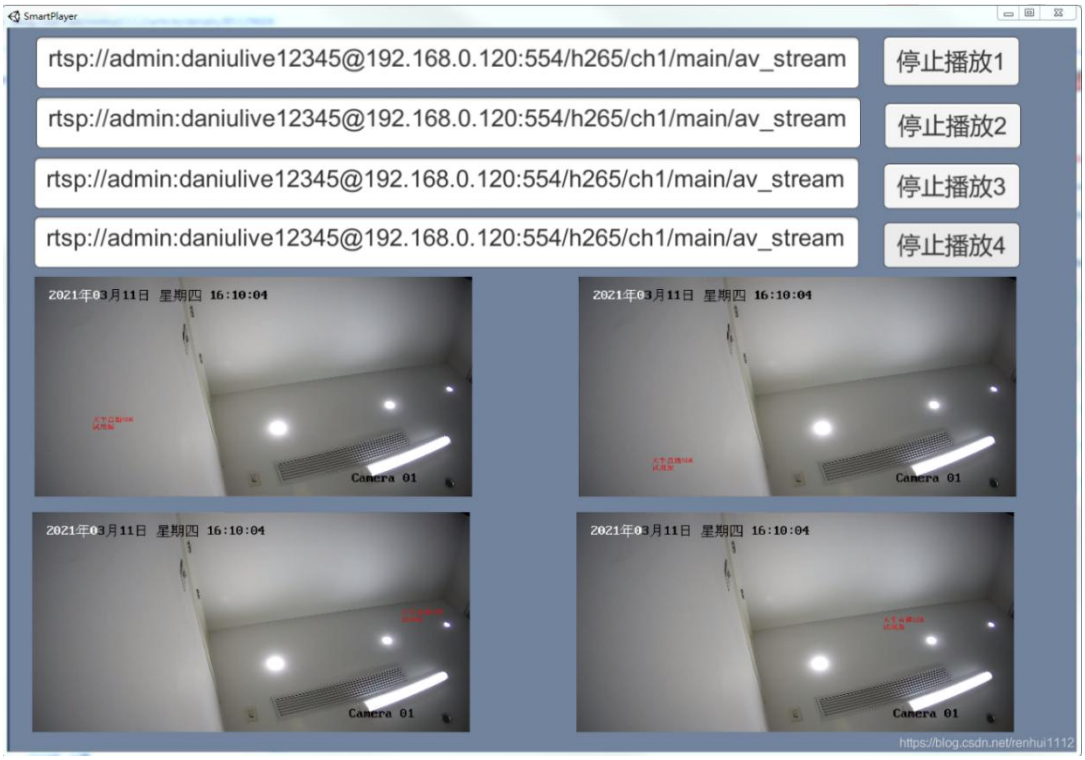
1. Windows 播放端 SDK 说明

随着 Unity3D 的应用范围越来越广，越来越多的行业开始基于 Unity3D 开发产品，如传统行业中虚拟仿真教育、航空工业、室内设计、城市规划、工业仿真等领域。

基于此，我们在 Unity 环境下推出了跨平台低延迟的 RTMP|RTSP 直播播放器，很好的解决了好多对延迟要求苛刻的使用场景。

1.1 demo 说明

- SmartU3dWinPlayer：大牛直播 SDK Unity3D Windows 平台 RTMP/RTSP 直播播放端工程。Demo 界面如下：



1.2 集成说明

- Unity3D 接口和调用 demo，参见：SmartPlayerWindowsMono.cs
- SmartU3dWinPlayer\Assets\Plugins\x86 和 SmartU3dWinPlayer\Assets\Plugins\x86_64 下相关库到工程：
- 将以下文件加到集成工程(SmartU3dWinPlayer\Assets\SmartPlayerSDK 目录下)：

- nt_base_code_define.cs；
- smart_player_define.cs；
- smart_player_sdk.cs；
- smart_log.cs；
- smart_log_define.cs。

➢ 如需集成到自己系统测试，请用大牛直播 SDK 的 app name(不然集成提示 license failed)，正式授权版按照授权 app name 正常使用即可，windows 测试 app name：SmartPlayer。

1.3 调用时序

Windows 平台 Unity3D 实际上调用的是大牛直播 SDK 的 C#相关接口实现，无需二次封装，相关调用时序，也可参照 Windows C#调用 SDK 调用说明(参见“视沃科技-Windows-SDK 集成说明.pdf”)。

调用流程：

1．准备播放：

开始播放的时候，调用 OpenPlayer()，完成 PlayerInit 和 PlayerOpen 操作，并设置 eventcallback 和 videoframecallback，并完成相关参数配置，设置 YUV 回调输出，调用 PlayerStart，完成播放端的启动。

注意：正式授权版，需在调用 NT_SP_Init()接口之前，调用 NT_SP_SetSDKClientKey()，设置相关 license key，确保验证通过。

```
public void Play(int sel)
 {
      if (videoctrl[sel].is_running)
      {
          Debug.Log("已经在播放..");
          return;
      }

      lock (videoctrl[sel].frame_lock_)
      {
          videoctrl[sel].cur_video_frame_ = null;
      }

      OpenPlayer(sel);

      if (videoctrl[sel].player_handle_ == IntPtr.Zero)
          return;

      //设置播放 URL
      NTSmartPlayerSDK.NT_SP_SetURL(videoctrl[sel].player_handle_, videoctrl[sel].videoUrl);

      /* ++ 播放前参数配置可加在此处 ++ */

      int play_buffer_time_ = 100;
      NTSmartPlayerSDK.NT_SP_SetBuffer(videoctrl[sel].player_handle_, play_buffer_time_);                    //设置 buffer
time
```

```
        int is_using_tcp = 0;           //TCP 模式

        NTSmartPlayerSDK.NT_SP_SetRTSPTcpMode(videoctrl[sel].player_handle_, is_using_tcp);

        int timeout = 10;
        NTSmartPlayerSDK.NT_SP_SetRtspTimeout(videoctrl[sel].player_handle_, timeout);

        int is_auto_switch_tcp_udp = 1;
        NTSmartPlayerSDK.NT_SP_SetRtspAutoSwitchTcpUdp(videoctrl[sel].player_handle_, is_auto_switch_tcp_udp);

        Boolean is_mute_ = false;
        NTSmartPlayerSDK.NT_SP_SetMute(videoctrl[sel].player_handle_, is_mute_ ? 1 : 0);                    //是否启动播
放的时候静音

        int is_fast_startup = 1;
        NTSmartPlayerSDK.NT_SP_SetFastStartup(videoctrl[sel].player_handle_, is_fast_startup);              //设置快速启
动模式

        Boolean is_low_latency_ = false;
        NTSmartPlayerSDK.NT_SP_SetLowLatencyMode(videoctrl[sel].player_handle_, is_low_latency_ ? 1 : 0);   //设置是否
启用低延迟模式

        //设置旋转角度(设置 0， 90， 180， 270 度有效，其他值无效)
        int rotate_degrees = 0;
        NTSmartPlayerSDK.NT_SP_SetRotation(videoctrl[sel].player_handle_, rotate_degrees);

            int volume = 100;
            NTSmartPlayerSDK.NT_SP_SetAudioVolume(videoctrl[sel].player_handle_, volume);   //设置播放音量，范围是[0,
100], 0 是静音，100 是最大音量，默认是 100


        // 设置上传下载报速度
        int is_report = 0;
        int report_interval = 1;
        NTSmartPlayerSDK.NT_SP_SetReportDownloadSpeed(videoctrl[sel].player_handle_, is_report, report_interval);
        /* -- 播放前参数配置可加在此处 -- */

        //video frame callback (YUV/RGB)
        videoctrl[sel].video_frame_call_back_ = new SP_SDKVideoFrameCallBack(NT_SP_SetVideoFrameCallBack);
        NTSmartPlayerSDK.NT_SP_SetVideoFrameCallBack(videoctrl[sel].player_handle_,
(Int32)NT.NTSmartPlayerDefine.NT_SP_E_VIDEO_FRAME_FORMAT.NT_SP_E_VIDEO_FRAME_FROMAT_I420,
window_handle_, videoctrl[sel].video_frame_call_back_);

        UInt32 flag = NTSmartPlayerSDK.NT_SP_StartPlay(videoctrl[sel].player_handle_);

        if (flag == DANIULIVE_RETURN_OK)
        {
            videoctrl[sel].is_need_get_frame_ = true;
            Debug.Log("播放成功");
        }
        else
        {
            videoctrl[sel].is_need_get_frame_ = false;
            Debug.LogError("播放失败");
        }

        videoctrl[sel].is_running = true;
    }
```

2. OpenPlayer 实现：

```
    private void OpenPlayer(int sel)
    {
        window_handle_ = IntPtr.Zero;
```

```
        if (videoctrl[sel].player_handle_ == IntPtr.Zero)
        {
            videoctrl[sel].player_handle_ = new IntPtr();
            UInt32 ret_open = NTSmartPlayerSDK.NT_SP_Open(out videoctrl[sel].player_handle_, window_handle_, 0,
IntPtr.Zero);
            if (ret_open != 0)
            {
                videoctrl[sel].player_handle_ = IntPtr.Zero;
                Debug.LogError("调用 NT_SP_Open 失败..");
                return;
            }
        }

        videoctrl[sel].event_call_back_ = new SP_SDKEventCallBack(NT_SP_SDKEventCallBack);
        NTSmartPlayerSDK.NT_SP_SetEventCallBack(videoctrl[sel].player_handle_, window_handle_,
videoctrl[sel].event_call_back_);

        videoctrl[sel].sdk_video_frame_call_back_ = new VideoControl.SetVideoFrameCallBack(SDKVideoFrameCallBack);
        videoctrl[sel].sdk_event_call_back_ = new VideoControl.SetEventCallBack(SDKEventCallBack);
    }
```

3. Video frame 实时处理并绘制：

开始播放后，daniulive 直播播放端 SDK 回调 yuv 数据及相关信息，unity3d 获取到数据信息后，调用 InitYUVTexture()，完成初始化工作，调用 UpdateYUVTexture()实现数据实时刷新，当数据信息发生变化时，会二次调用 InitYUVTexture()，完成初始化。

```
    private void SDKVideoFrameCallBack(UInt32 status, IntPtr frame, int sel)
    {
        //这里拿到回调 frame，进行相关操作
        NT_SP_VideoFrame video_frame = (NT_SP_VideoFrame)Marshal.PtrToStructure(frame, typeof(NT_SP_VideoFrame));

        VideoFrame   u3d_frame = new VideoFrame();

        u3d_frame.width_   = video_frame.width_;
        u3d_frame.height_ = video_frame.height_;

        u3d_frame.timestamp_ = (UInt64)video_frame.timestamp_;

        int d_y_stride = video_frame.width_;
        int d_u_stride = (video_frame.width_ + 1) / 2;
        int d_v_stride = d_u_stride;

        int d_y_size = d_y_stride * video_frame.height_;
        int d_u_size = d_u_stride * ((video_frame.height_ + 1) / 2);
        int d_v_size = d_u_size;

        int u_v_height = ((u3d_frame.height_ + 1) / 2);

        u3d_frame.y_stride_ = d_y_stride;
        u3d_frame.u_stride_ = d_u_stride;
        u3d_frame.v_stride_ = d_v_stride;

        u3d_frame.y_data_ = new byte[d_y_size];
        u3d_frame.u_data_ = new byte[d_u_size];
        u3d_frame.v_data_ = new byte[d_v_size];


        CopyFramePlane(u3d_frame.y_data_, d_y_stride,
            video_frame.plane0_, video_frame.stride0_, u3d_frame.height_);

        CopyFramePlane(u3d_frame.u_data_, d_u_stride,
            video_frame.plane1_, video_frame.stride1_, u_v_height);
```

```
        CopyFramePlane(u3d_frame.v_data_, d_v_stride,
            video_frame.plane2_, video_frame.stride2_, u_v_height);

        lock (videoctrl[sel].frame_lock_ )
        {
            videoctrl[sel].cur_video_frame_ = u3d_frame;
            //Debug.LogError("sel: " + sel + " w:" + u3d_frame.width_ + "h:" + u3d_frame.height_);
        }
    }
```

4．停止播放

调用 NT_U3D_StopPlay()后，调用 NT_U3D_UnInit()即可。

```
    private void ClosePlayer(int sel)
    {
        videoctrl[sel].is_need_get_frame_ = false;
        videoctrl[sel].is_need_init_texture_ = false;

        if (videoctrl[sel].player_handle_ == IntPtr.Zero)
        {
            return;
        }

        UInt32 flag = NTSmartPlayerSDK.NT_SP_StopPlay(videoctrl[sel].player_handle_);
        if (flag == DANIULIVE_RETURN_OK)
        {
            Debug.Log("停止成功");
        }
        else
        {
            Debug.LogError("停止失败");
        }

        videoctrl[sel].player_handle_ = IntPtr.Zero;

        videoctrl[sel].is_running = false;
    }
```

5．Event 回调

```
    private void SDKEventCallBack(UInt32 event_id,
        Int64 param1,
        Int64 param2,
        UInt64 param3,
        [MarshalAs(UnmanagedType.LPStr)] String param4,
        [MarshalAs(UnmanagedType.LPStr)] String param5,
        IntPtr param6,
        int sel)
    {
        if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_CONNECTING == event_id
            || (UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_CONNECTION_FAILED == event_id
            || (UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_CONNECTED == event_id
            || (UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_DISCONNECTED == event_id)
        {
            connection_status_ = event_id;
        }
        if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_START_BUFFERING == event_id
            || (UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_BUFFERING == event_id
            || (UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_STOP_BUFFERING == event_id)
        {
            buffer_status_ = event_id;
            if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_BUFFERING == event_id)
            {
                buffer_percent_ = (Int32)param1;
```

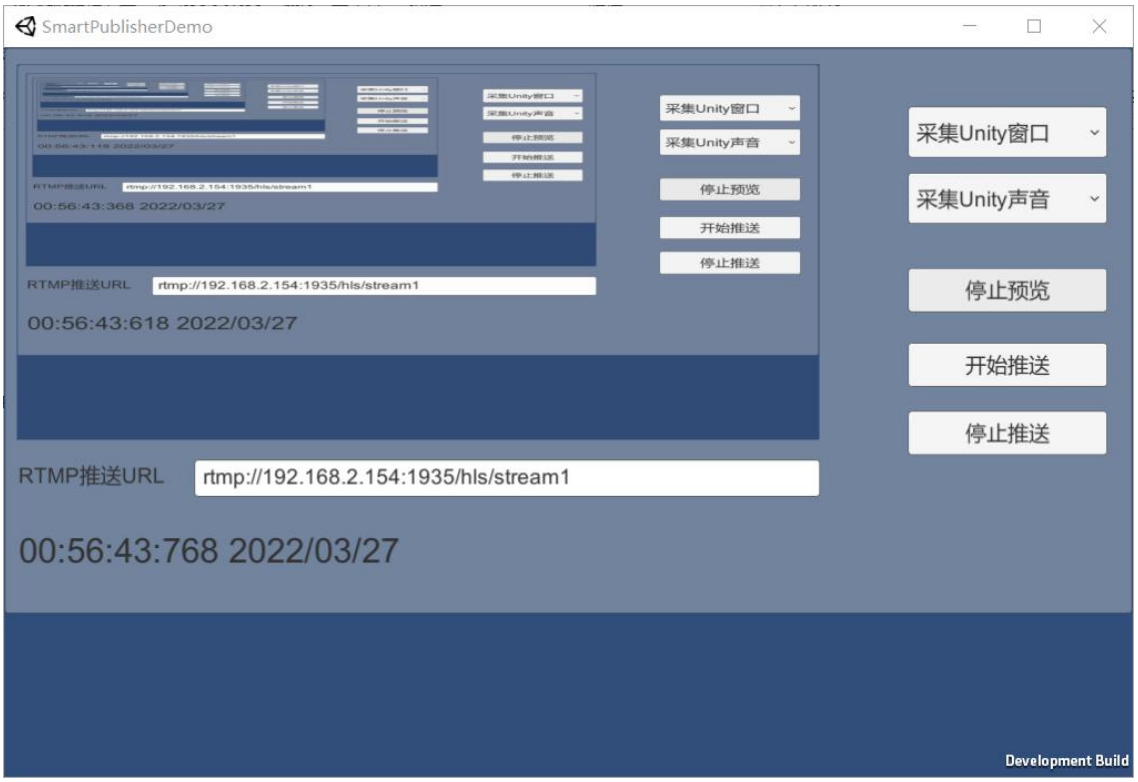
```
            }
        }
        if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_DOWNLOAD_SPEED == event_id)
        {
            download_speed_ = (Int32)param1;
        }

        String t_show_str = "";
        if (connection_status_ != 0)
        {
            t_show_str += "链接状态: ";
            if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_CONNECTING ==
connection_status_)
            {
                t_show_str += "链接中";
            }
            else if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_CONNECTION_FAILED ==
connection_status_)
            {
                t_show_str += "链接失败";
            }
            else if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_CONNECTED ==
connection_status_)
            {
                t_show_str += "链接成功";
            }
            else if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_DISCONNECTED ==
connection_status_)
            {
                t_show_str += "链接断开";
            }
        }

        if (download_speed_ != -1)
        {
            String ss = "    下载速度: " + (download_speed_ * 8 / 1000).ToString() + "kbps " + (download_speed_ / 1024).ToString()
+ "KB/s";
            t_show_str += ss;
        }

        if (buffer_status_ != 0)
        {
             t_show_str += "    缓冲状态: ";
            if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_START_BUFFERING ==
buffer_status_)
            {
                t_show_str += "开始缓冲";
            }
            else if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_BUFFERING == buffer_status_)
            {
                String ss = "缓冲中  " + buffer_percent_.ToString() + "%";
                t_show_str += ss;
            }
            else if ((UInt32)NTSmartPlayerDefine.NT_SP_E_EVENT_ID.NT_SP_E_EVENT_ID_STOP_BUFFERING ==
buffer_status_)
            {
                t_show_str += "结束缓冲";
            }
        }
        //show_message = "DaniuEvent: instance" + sel + ": " + t_show_str;
        //Debug.LogError(show_message);
    }
```

2. Windows 推送端 SDK 说明

Windows 平台 Unity3D 下的 RTMP 直播推送 SDK，数据源可以是 Unity 的窗口、摄像头或整个屏幕，音频可以是 Unity 的 AudioClip 音频、麦克风、扬声器，编码传输模块，还是调用官方原生接口。

2.1 demo 说明

SmartU3dWinPublisher: 大牛直播 SDK Unity3D Windows 平台 RTMP 直播推送 SDK Demo 工程，Demo 界面如下：

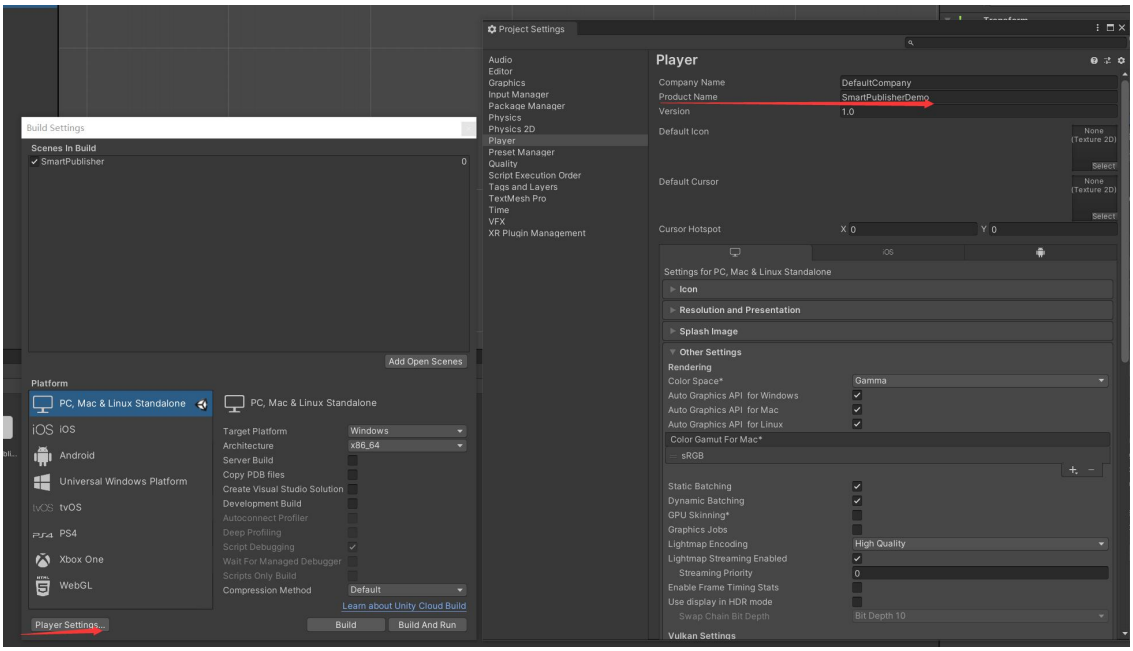


2.2 集成说明

- Unity3D 接口和调用 demo，参见：SmartPublishWinMono.cs
- SmartU3dWinPublisher\Assets\Plugins\x86 和 SmartU3dWinPublisher\Assets\Plugins\x86_64 下相关库到工程：
- 添加以下文件到集成工程(SmartU3dWinPublisher\Assets\SmartPublisherSDK 下)：
  - nt_base_code_define.cs；
  - nt_common_media_define.cs；

- nt_publisher_wrapper.cs；
- nt_smart_publisher_define.cs；
- nt_smart_publisher_sdk.cs；
- smart_log.cs；
- smart_log_define.cs。

➢ 如需集成到自己系统测试，请用大牛直播 SDK 的 app name(不然集成提示 license failed)，正式授权版按照授权 app name 正常使用即可，windows 测试 app name:SmartPublisherDemo。



2.3 调用时序

Windows 平台 Unity3D 实际上调用的是大牛直播 SDK 的 C#相关接口实现，无需二次封装，相关调用时序，也可参照 Windows C#调用 SDK 调用说明(参见“视沃科技-Windows-SDK 集成说明.pdf”)。

1. 基础初始化

```
private bool InitSDK()
{
    if (!is_pusher_sdk_init_)
    {
        // 设置日志路径(请确保目录存在)
        String log_path = "D:\\pulisherlog";
        NTSmartLog.NT_SL_SetPath(log_path);

        UInt32 isInited = NTSmartPublisherSDK.NT_PB_Init(0, IntPtr.Zero);

        if (isInited != 0)
        {
            Debug.Log("调用 NT_PB_Init 失败..");
```

```
                return false;
            }

            is_pusher_sdk_init_ = true;
        }

        return true;
    }
```

2．调用 Open()接口，获取推送实例

```
public bool OpenPublisherHandle(uint video_option, uint audio_option)
    {
        if (publisher_handle_ != IntPtr.Zero)
        {
            return true;
        }

        publisher_handle_count_ = 0;

        if (NTBaseCodeDefine.NT_ERC_OK != NTSmartPublisherSDK.NT_PB_Open(out publisher_handle_,
            video_option, audio_option, 0, IntPtr.Zero))
        {
            return false;
        }

        if (publisher_handle_ != IntPtr.Zero)
        {
            pb_event_call_back_ = new NT_PB_SDKEventCallBack(PbEventCallBack);

            NTSmartPublisherSDK.NT_PB_SetEventCallBack(publisher_handle_, IntPtr.Zero, pb_event_call_back_);

            return true;
        }
        else
        {
            return false;
        }
    }
```

3．初始化参数配置

这里需要注意下，如果要采集 unity 窗口，需要设置图层模式，先填充一层 RGBA 黑色背景，然后再添加一层，用于叠加外部数据。

```
private void SetCommonOptionToPublisherSDK()
    {
        if (!IsPublisherHandleAvailable())
        {
            Debug.Log("SetCommonOptionToPublisherSDK, publisher handle with null..");
            return;
        }

        NTSmartPublisherSDK.NT_PB_ClearLayersConfig(publisher_handle_, 0,
                        0, IntPtr.Zero);

        if (video_option == NTSmartPublisherDefine.NT_PB_E_VIDEO_OPTION.NT_PB_E_VIDEO_OPTION_LAYER)
        {
            // 第 0 层填充 RGBA 矩形，目的是保证帧率，颜色就填充全黑
            int red = 0;
            int green = 0;
            int blue = 0;
            int alpha = 255;
```

```
                NT_PB_RGBARectangleLayerConfig rgba_layer_c0 = new NT_PB_RGBARectangleLayerConfig();

                rgba_layer_c0.base_.type_ =
(Int32)NTSmartPublisherDefine.NT_PB_E_LAYER_TYPE.NT_PB_E_LAYER_TYPE_RGBA_RECTANGLE;
                rgba_layer_c0.base_.index_ = 0;
                rgba_layer_c0.base_.enable_ = 1;
                rgba_layer_c0.base_.region_.x_ = 0;
                rgba_layer_c0.base_.region_.y_ = 0;
                rgba_layer_c0.base_.region_.width_ = video_width_;
                rgba_layer_c0.base_.region_.height_ = video_height_;

                rgba_layer_c0.base_.offset_ = Marshal.OffsetOf(rgba_layer_c0.GetType(), "base_").ToInt32();
                rgba_layer_c0.base_.cb_size_ = (uint)Marshal.SizeOf(rgba_layer_c0);

                rgba_layer_c0.red_ = System.BitConverter.GetBytes(red)[0];
                rgba_layer_c0.green_ = System.BitConverter.GetBytes(green)[0];
                rgba_layer_c0.blue_ = System.BitConverter.GetBytes(blue)[0];
                rgba_layer_c0.alpha_ = System.BitConverter.GetBytes(alpha)[0];

                IntPtr rgba_conf = Marshal.AllocHGlobal(Marshal.SizeOf(rgba_layer_c0));

                Marshal.StructureToPtr(rgba_layer_c0, rgba_conf, true);

                UInt32 rgba_r = NTSmartPublisherSDK.NT_PB_AddLayerConfig(publisher_handle_, 0,
                                rgba_conf,
(int)NTSmartPublisherDefine.NT_PB_E_LAYER_TYPE.NT_PB_E_LAYER_TYPE_RGBA_RECTANGLE,
                                0, IntPtr.Zero);

                Marshal.FreeHGlobal(rgba_conf);

                NT_PB_ExternalVideoFrameLayerConfig external_layer_c1 = new NT_PB_ExternalVideoFrameLayerConfig();

                external_layer_c1.base_.type_ =
(Int32)NTSmartPublisherDefine.NT_PB_E_LAYER_TYPE.NT_PB_E_LAYER_TYPE_EXTERNAL_VIDEO_FRAME;
                external_layer_c1.base_.index_ = 1;
                external_layer_c1.base_.enable_ = 1;
                external_layer_c1.base_.region_.x_ = 0;
                external_layer_c1.base_.region_.y_ = 0;
                external_layer_c1.base_.region_.width_ = video_width_;
                external_layer_c1.base_.region_.height_ = video_height_;

                external_layer_c1.base_.offset_ = Marshal.OffsetOf(external_layer_c1.GetType(), "base_").ToInt32();
                external_layer_c1.base_.cb_size_ = (uint)Marshal.SizeOf(external_layer_c1);

                IntPtr external_layer_conf = Marshal.AllocHGlobal(Marshal.SizeOf(external_layer_c1));

                Marshal.StructureToPtr(external_layer_c1, external_layer_conf, true);

                UInt32 external_r = NTSmartPublisherSDK.NT_PB_AddLayerConfig(publisher_handle_, 0,
                                external_layer_conf,
(int)NTSmartPublisherDefine.NT_PB_E_LAYER_TYPE.NT_PB_E_LAYER_TYPE_EXTERNAL_VIDEO_FRAME,
                                0, IntPtr.Zero);

                Marshal.FreeHGlobal(external_layer_conf);

            }
            else if (video_option ==
NTSmartPublisherDefine.NT_PB_E_VIDEO_OPTION.NT_PB_E_VIDEO_OPTION_CAMERA)
            {
                CameraInfo camera = cameras_[cur_sel_camera_index_];
                NT_PB_VideoCaptureCapability cap = camera.capabilities_[cur_sel_camera_resolutions_index_];

                SetVideoCaptureDeviceBaseParameter(camera.id_.ToString(), (UInt32)cap.width_, (UInt32)cap.height_);
            }
```

```
        SetFrameRate((uint)video_fps_);

        Int32 type = 0;     //软编码
        Int32 encoder_id = 1;
        UInt32 codec_id = (UInt32)NTCommonMediaDefine.NT_MEDIA_CODEC_ID.NT_MEDIA_CODEC_ID_H264;
        Int32 param1 = 0;

        SetVideoEncoder(type, encoder_id, codec_id, param1);

        SetVideoQualityV2(CalVideoQuality(video_width_, video_height_, is_h264_encoder));

        SetVideoBitRate(CalBitRate(video_fps_, video_width_, video_height_));

        SetVideoMaxBitRate((CalMaxKBitRate(video_fps_, video_width_, video_height_, false)));

        SetVideoKeyFrameInterval((edit_key_frame_));
        if (is_h264_encoder)
        {
            SetVideoEncoderProfile(1);
        }

        SetVideoEncoderSpeed(CalVideoEncoderSpeed(video_width_, video_height_, is_h264_encoder));

        // 音频相关设置

        SetAuidoInputDeviceId(0);
        SetPublisherAudioCodecType(1);
        SetPublisherMute(is_mute);
        SetEchoCancellation(0, 0);
        SetNoiseSuppression(0);
        SetAGC(0);
        SetVAD(0);
        SetInputAudioVolume(Convert.ToSingle(edit_audio_input_volume_));

    }
```

4．数据采集

摄像头和屏幕的数据采集，还是调用原生的 SDK 接口，本文不再赘述，如果需要采集 Unity 窗体的数据，可以用参考以下代码：

```
    if ( texture_ == null || video_width_ != Screen.width || video_height_ != Screen.height)
    {
        Debug.Log("OnPostRender screen changed++ scr_width: " + Screen.width + " scr_height: " + Screen.height);

        if (screen_image_ != IntPtr.Zero)
        {
            Marshal.FreeHGlobal(screen_image_);
            screen_image_ = IntPtr.Zero;
        }

        if (texture_ !=  null)
        {
            UnityEngine.Object.Destroy(texture_);
            texture_ = null;
        }

        video_width_ = Screen.width;
        video_height_ = Screen.height;
```

```
            texture_ = new Texture2D(video_width_, video_height_, TextureFormat.BGRA32, false);

            screen_image_ = Marshal.AllocHGlobal(video_width_ * 4 * video_height_);

            Debug.Log("OnPostRender screen changed--");

            return;
        }

        texture_.ReadPixels(new Rect(0, 0, video_width_, video_height_), 0, 0, false);
        texture_.Apply();
```

从 texture 里面，通过调用 GetRawTextureData()，获取到原始数据。

如需采集 Unity 的 AudioClip 数据：

```
/// <summary>
///  获取 AudioClip 数据
/// </summary>
private void PostUnityAudioClipData()
{
    TimerManager.UnRegister(this);

    audio_clip_info_ = new AudioClipInfo();

    audio_clip_info_.audio_source_ = GameObject.Find("Canvas/Panel/AudioSource").GetComponent<AudioSource>();

    if (audio_clip_info_.audio_source_ == null)
    {
        Debug.LogError("audio source is null..");
        return;
    }

    if (audio_clip_info_.audio_source_.clip == null)
    {
        Debug.LogError("audio clip is null..");
        return;
    }

    audio_clip_info_.audio_clip_ = audio_clip_info_.audio_source_.clip;
    audio_clip_info_.audio_clip_offset_ = 0;
    audio_clip_info_.audio_clip_length_ = audio_clip_info_.audio_clip_.samples * audio_clip_info_.audio_clip_.channels;

    pcm_mute_data_ = FillPcmMuteData(audio_clip_info_.audio_clip_);

    TimerManager.Register(this, 0.019999f, (PostPCMData), false);
}
```

5. 数据对接

Unity 的视频数据，获取到 Texture 数据后，通过调用 OnPostRGBAData()接口，传递给 SDK 层。

如果是 Unity 的 AudioClip 采集的数据，调用 OnPostAudioPCMFloatData()传递给 SDK。

```
publisher_wrapper_.OnPostAudioPCMFloatData(pcm_data.data_,
    pcm_data.size_,
    pcm_time_stamp_,
    pcm_data.sample_rate_,
    pcm_data.channels_,
    pcm_data.per_channel_sample_number_);
```

6. 本地数据预览

```
public bool StartPreview()
{
    if(CheckPublisherHandleAvailable() == false)
        return false;

    video_preview_image_callback_ = new
NT_PB_SDKVideoPreviewImageCallBack(SDKVideoPreviewImageCallBack);

    NTSmartPublisherSDK.NT_PB_SetVideoPreviewImageCallBack(publisher_handle_,
(int)NTSmartPublisherDefine.NT_PB_E_IMAGE_FORMAT.NT_PB_E_IMAGE_FORMAT_RGB32, IntPtr.Zero,
video_preview_image_callback_);

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

        return false;
    }

    publisher_handle_count_++;

    is_previewing_ = true;

    return true;
}

public void StopPreview()
{
    if (is_previewing_ == false) return;

    is_previewing_ = false;

    publisher_handle_count_--;
    NTSmartPublisherSDK.NT_PB_StopPreview(publisher_handle_);

    if (0 == publisher_handle_count_)
    {
        NTSmartPublisherSDK.NT_PB_Close(publisher_handle_);
        publisher_handle_ = IntPtr.Zero;
    }
}
```

设置 preview 后，处理 preview 的数据回调。

```
//预览数据回调
public void SDKVideoPreviewImageCallBack(IntPtr handle, IntPtr user_data, IntPtr image)
{
    NT_PB_Image pb_image = (NT_PB_Image)Marshal.PtrToStructure(image, typeof(NT_PB_Image));

    NT_VideoFrame pVideoFrame = new NT_VideoFrame();

    pVideoFrame.width_ = pb_image.width_;
    pVideoFrame.height_ = pb_image.height_;

    pVideoFrame.stride_ = pb_image.stride_[0];

    Int32 argb_size = pb_image.stride_[0] * pb_image.height_;

    pVideoFrame.plane_data_ = new byte[argb_size];
```

```
        if (argb_size > 0)
        {
            Marshal.Copy(pb_image.plane_[0],pVideoFrame.plane_data_,0, argb_size);
        }

        {
            cur_image_ = pVideoFrame;
        }

    }
```

7．相关 event 回调处理

```
    private void PbEventCallBack(IntPtr handle, IntPtr user_data,
        UInt32 event_id,
        Int64 param1,
        Int64 param2,
        UInt64 param3,
        UInt64 param4,
        [MarshalAs(UnmanagedType.LPStr)] String param5,
        [MarshalAs(UnmanagedType.LPStr)] String param6,
        IntPtr param7)
    {
        String event_log = "";

        switch (event_id)
        {
            case (uint)NTSmartPublisherDefine.NT_PB_E_EVENT_ID.NT_PB_E_EVENT_ID_CONNECTING:
                event_log = "连接中";
                if (!String.IsNullOrEmpty(param5))
                {
                    event_log = event_log + " url:" + param5;
                }
                break;

            case (uint)NTSmartPublisherDefine.NT_PB_E_EVENT_ID.NT_PB_E_EVENT_ID_CONNECTION_FAILED:
                event_log = "连接失败";
                if (!String.IsNullOrEmpty(param5))
                {
                    event_log = event_log + " url:" + param5;
                }
                break;

            case (uint)NTSmartPublisherDefine.NT_PB_E_EVENT_ID.NT_PB_E_EVENT_ID_CONNECTED:
                event_log = "已连接";
                if (!String.IsNullOrEmpty(param5))
                {
                    event_log = event_log + " url:" + param5;
                }
                break;

            case (uint)NTSmartPublisherDefine.NT_PB_E_EVENT_ID.NT_PB_E_EVENT_ID_DISCONNECTED:
                event_log = "断开连接";
                if (!String.IsNullOrEmpty(param5))
                {
                    event_log = event_log + " url:" + param5;
                }
                break;

            default:
                break;
        }
```

```
        if(OnLogEventMsg != null) OnLogEventMsg.Invoke(event_id, event_log);
    }
```

8．开始推送、停止推送

```
    public bool StartPublisher(String url)
    {
        if (CheckPublisherHandleAvailable() == false) return false;

        if (publisher_handle_ == IntPtr.Zero)
        {
            return false;
        }
        if (!String.IsNullOrEmpty(url))
        {
            NTSmartPublisherSDK.NT_PB_SetURL(publisher_handle_, url, IntPtr.Zero);
        }

        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;

            return false;
        }

        publisher_handle_count_++;

        is_publishing_ = true;

        return true;
    }

    public void StopPublisher()
    {
        if (is_publishing_ == false) return;

        publisher_handle_count_--;
        NTSmartPublisherSDK.NT_PB_StopPublisher(publisher_handle_);

        if (0 == publisher_handle_count_)
        {
            NTSmartPublisherSDK.NT_PB_Close(publisher_handle_);
            publisher_handle_ = IntPtr.Zero;
        }

        is_publishing_ = false;
    }
```

9．关闭实例

```
    public void Close()
    {
        if (0 == publisher_handle_count_)
        {
            NTSmartPublisherSDK.NT_PB_Close(publisher_handle_);
            publisher_handle_ = IntPtr.Zero;
        }
    }
```

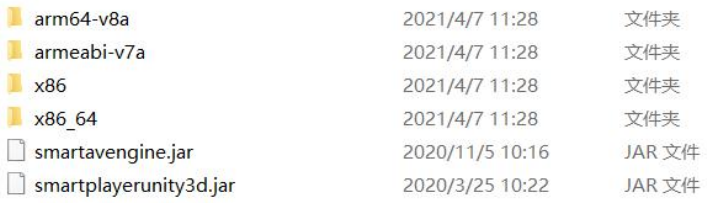
3. Android 播放端 SDK 说明

3.1 demo 说明

- SmartU3dAndroidPlayer：大牛直播 SDK Unity3D Android 平台 RTMP/RTSP 直播播放端工程。

3.2 集成说明

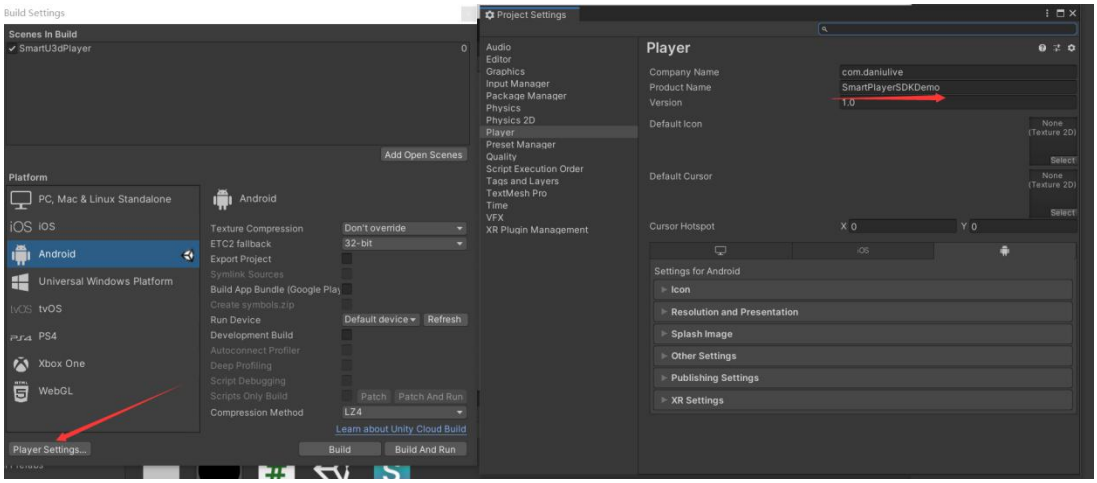
- Unity3D 接口和调用 demo，参见：SmartPlayerAndroidMono.cs
- SmartU3dAndroidPlayer\Assets\Plugins\Android\libs 下相关库到工程：



- smartavengine.jar 加入到工程；
- smartplayerunity3d.jar 加入工程；
- 32/64 位 arm 和 x86 下 libSmartPlayer.so；
- 在 SmartU3dAndroidPlayer\Assets\Plugins\Android\AndroidManifest.xml 配置相关权限：

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" >
</uses-permission>
<uses-permission android:name="android.permission.INTERNET" ></uses-permission>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
```

- 如需集成到自己系统测试，请用大牛直播 SDK 的 app name(不然集成提示 license failed)，正式授权版按照授权 app name 正常使用即可；
- 正式授权版，需要调用 license 设置接口，确保 license key 正确设置方可正常使用；
- 如何改 app-name：
  Unity3D 模式下：File-->Build Settings-->Android-->Player Settings:



3.3 调用时序

1. **【最先调用】**NT_U3D_Init：player 初始化，目前预留；

2. **【正式授权版设置 license key】**NT_U3D_SetSDKClientKey，设置正式授权版 license，请早于 NT_U3D_Open 接口调用；

3. **【获得 player 句柄】**NT_U3D_Open，设置上下文信息，返回 player 句柄；

4. 【设置 GameObject】NT_U3D_Set_Game_Object，注册 Game Object，用于消息传递；

5. 【设置硬解码】NT_U3D_SetVideoDecoderMode，设置特定机型硬解码播放，如硬解码检测到不支持，切换到软解码；

6. 【audio 输出类型】NT_U3D_SetAudioOutputType()，如果 use_audiotrack 设置为 0，将会自动选择输出设备，如果设置为 1，使用 audiotrack 模式，一般建议使用 audiotrack 模式；

7. 【缓冲设置】NT_U3D_SetBuffer，设置播放端缓存数据 buffer，以毫秒(ms)为单位，如超低延迟模式下，不需 buffer 数据，设置为 0；

8. 【RTSP TCP/UDP 设置】NT_U3D_SetRTSPTcpMode，设置 TCP/UDP 播放模式，**注意：此接口仅用于 RTSP；**

9. 【RTSP 超时时间设置】NT_U3D_SetRTSPTimeout，设置 RTSP 超时时间, timeout 单位为秒，必须大于 0；

10. 【RTSP TCP/UDP 自动切换】NT_U3D_SetRTSPAutoSwitchTcpUdp，设置 RTSP TCP/UDP 自动切换，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；

11. **【实时静音-可实时调用】**NT_U3D_SetMute，设置播放过程中，实时静音/取消静音；

12. 【快速启动】NT_U3D_SetFastStartup，设置快速启动后，如果 CDN 缓存 GOP，播放端可快速出帧；

13. 【低延迟模式】NT_U3D_SetPlayerLowLatencyMode，设置低延迟模式；

14. **【视频垂直反转-可实时调用】**NT_U3D_SetFlipVertical，**视频垂直反转；**

15. **【视频水平反转-可实时调用】**NT_U3D_SetFlipHorizontal，**视频水平反转；**

16. **【视频显示角度设置-可实时调用】**NT_U3D_SetRotation，**针对类似于安防摄像头或其他设备出来的图像倒置现象，支持视频播放 view 顺时针旋转，当前支持 0 度，90 度, 180 度, 270 度 旋转，注意除了 0 度之外， 其他角度都会额外消耗性能；**

17. 【下载速度回调设置】NT_U3D_SetReportDownloadSpeed，设置下载速度上报，默认不上报下载速度；

18. 【快照设置】NT_U3D_SetSaveImageFlag，设置是否需要在播放或录像过程中快照；

19. **【快照-录像或播放后，可随时调用】**NT_U3D_SaveCurImage，播放过程中，根据设置路径和文件名，实时快照；

20. **【快速切换 url-可实时调用】**NT_U3D_SwitchPlaybackUrl，快速切换播放 url，快速切换时，只换播放 source 部分，适用于不同数据流之间，快速切换；

21. 【录像设置】NT_U3D_CreateFileDirectory，创建文件路径；

22. 【录像设置】NT_U3D_SetRecorderDirectory，设置文件路径；

23. 【录像设置】NT_U3D_SetRecorderFileMaxSize，设置每个录像文件最大 size，以兆（M）为单位，范围[5M~500M]；

24. 【录像设置】NT_U3D_SetRecorderAudioTranscodeAAC，支持拉取的 RTMP/RTSP 的 PCMA/PCMU/SPEEX 音频格式转 AAC 后录制；

25. 【设置播放或录像 URL】NT_U3D_SetUrl，设置播放/录像 url；

26. **【播放】**NT_U3D_StartPlay，开始播放；

27. **【播放】**NT_U3D_GetVideoFrame，获取底层回调的 YUV 数据；

28. **【播放】**NT_U3D_StopPlay，停止播放；

29. **【录像】**NT_U3D_StartRecorder，开始录像；

30. **【录像】**NT_U3D_StopRecorder，停止录像；

31. **【关闭】**NT_U3D_Close，关闭播放器实例；

32. **【最后调用】**NT_U3D_UnInit，UnInit Player，最后调用。

3.4 相关实现

1. 初始化播放

完成相关的初始化和接口参数设定后，调用 NT_U3D_StartPlay()实现音视频数据播放。

```
public void Play(int sel)
{
    if (java_obj_cur_activity_ == null)
    {
        Debug.LogError("getApplicationContext is null");
        return;
    }

    videoctrl[sel].player_handle_ = NT_U3D_Open();
    if (videoctrl[sel].player_handle_ == 0)
    {
        Debug.LogError("open fail");
        return;
    }

    NT_U3D_Set_Game_Object(videoctrl[sel].player_handle_, game_object_);

    NT_U3D_SetUrl(videoctrl[sel].player_handle_, videoctrl[sel].videoUrl);

    /* ++ 播放前参数配置可加在此处 ++ */
    NT_U3D_SetRTSPTcpMode(videoctrl[sel].player_handle_, 0);

    NT_U3D_SetReportDownloadSpeed(videoctrl[sel].player_handle_, 0, 1);//下载速度回调

    NT_U3D_SetBuffer(videoctrl[sel].player_handle_, 100);//设置 buffer time

    NT_U3D_SetPlayerLowLatencyMode(videoctrl[sel].player_handle_, 0);//设置是否启用低延迟模式

    NT_U3D_SetMute(videoctrl[sel].player_handle_, 0);//是否启动播放的时候静音

    int cur_audio_volume = 100;             //默认播放音量
    NT_U3D_SetAudioVolume(videoctrl[sel].player_handle_, cur_audio_volume);                 //设置播放音量

    int is_hw_decode = 1;
    NT_U3D_SetVideoDecoderMode(videoctrl[sel].player_handle_, is_hw_decode);                //设置 H.264 软硬解模式

    NT_U3D_SetVideoHevcDecoderMode(videoctrl[sel].player_handle_, is_hw_decode);            //设置 H.265 软硬解模式

    NT_U3D_SetFastStartup(videoctrl[sel].player_handle_, 1);//设置快速启动模式

    NT_U3D_SetRTSPTimeout(videoctrl[sel].player_handle_, 10);//设置 RTSP 超时时间

    NT_U3D_SetRTSPAutoSwitchTcpUdp(videoctrl[sel].player_handle_, 1);//设置 TCP/UDP 模式自动切换
    int is_audiotrack = 1;
    NT_U3D_SetAudioOutputType(videoctrl[sel].player_handle_, is_audiotrack);                //设置音频输出模式: if
0: 自动选择; if with 1: audiotrack 模式
    /* -- 播放前参数配置可加在此处 -- */

    int flag = NT_U3D_StartPlay(videoctrl[sel].player_handle_);

    if (flag   == DANIULIVE_RETURN_OK)
    {
        videoctrl[sel].is_need_get_frame_ = true;
        Debug.Log("播放成功");
    }
    else
    {
        videoctrl[sel].is_need_get_frame_ = false;
        Debug.LogError("播放失败");
    }
```

```
        videoctrl[sel].is_running = true;
    }
```

2. 停止播放：

调用 NT_U3D_StopPlay()后，如果没有录像，调用 NT_U3D_Close()，播放 handler 置空，然后调用 NT_U3D_UnInit()即可。

```
    private void ClosePlayer(int sel)
    {
        videoctrl[sel].is_need_get_frame_ = false;
        videoctrl[sel].is_need_init_texture_ = false;

        int flag = NT_U3D_StopPlay(videoctrl[sel].player_handle_);
        if (flag == DANIULIVE_RETURN_OK)
        {
            Debug.Log("停止成功");
        }
        else
        {
            Debug.LogError("停止失败");
        }

        flag = NT_U3D_Close(videoctrl[sel].player_handle_);
        if (flag == DANIULIVE_RETURN_OK)
        {
            Debug.Log("关闭成功");
        }
        else
        {
            Debug.LogError("关闭失败");
        }

        videoctrl[sel].player_handle_ = 0;

        videoctrl[sel].video_width_ = 0;
        videoctrl[sel].video_height_ = 0;
        videoctrl[sel].is_running = false;
    }
```

3. 相关 Event 处理：

```
    /// <summary>
    /// android 传递过来 code
    /// </summary>
    /// <param name="code"></param>
    public void onNTSmartEvent(string param)
    {
        if (!param.Contains(","))
        {
            Debug.Log("[onNTSmartEvent] android 传递参数错误");
            return;
        }

       string[] strs = param.Split(',');

       string player_handle =strs[0];
       string code = strs[1];
       string param1 = strs[2];
       string param2 = strs[3];
       string param3 = strs[4];
       string param4 = strs[5];
```

```
Debug.Log("[onNTSmartEvent] code: 0x" + Convert.ToString(Convert.ToInt32(code), 16));

 switch (Convert.ToInt32(code))
 {
     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_STARTED:
         Debug.Log("开始。。");
         break;
     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_CONNECTING:
         Debug.Log("连接中。。");
         break;
     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_CONNECTION_FAILED:
         Debug.Log("连接失败。。");
         break;
     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_CONNECTED:
         Debug.Log("连接成功。。");
         break;
     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_DISCONNECTED:
         Debug.Log("连接断开。。");
         break;
     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_STOP:
         Debug.Log("停止播放。。");
         break;
     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_RESOLUTION_INFO:
         Debug.Log("分辨率信息: width: " + Convert.ToInt32(param1) + ", height: " + Convert.ToInt32(param2));
         break;
     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_NO_MEDIADATA_RECEIVED:
         Debug.Log("收不到媒体数据，可能是 url 错误。。");
         break;
     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_SWITCH_URL:
         Debug.Log("切换播放 URL。。");
         break;

     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_CAPTURE_IMAGE:
         Debug.Log("快照: " + param1 + " 路径：" + param3);

         if (Convert.ToInt32(param1) == 0)
         {
             Debug.Log("截取快照成功。.");
         }
         else
         {
             Debug.Log("截取快照失败。.");
         }
         break;

     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_RECORDER_START_NEW_FILE:
         Debug.Log("[record]开始一个新的录像文件 ：" + param3);
         break;
     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_ONE_RECORDER_FILE_FINISHED:
         Debug.Log("[record]已生成一个录像文件 ：" + param3);
         break;

     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_START_BUFFERING:
         Debug.Log("Start_Buffering");
         break;

     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_BUFFERING:
         Debug.Log("Buffering: " + Convert.ToInt32(param1));
         break;

     case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_STOP_BUFFERING:
         Debug.Log("Stop_Buffering");
         break;
```

```
            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_DOWNLOAD_SPEED:
                Debug.Log("download_speed:" + param1 + "Byte/s" + ", "
                        + (Convert.ToInt32(param1) * 8 / 1000) + "kbps" + ", " + (Convert.ToInt32(param1) / 1024)
                        + "KB/s");
                break;
        }

    }
```

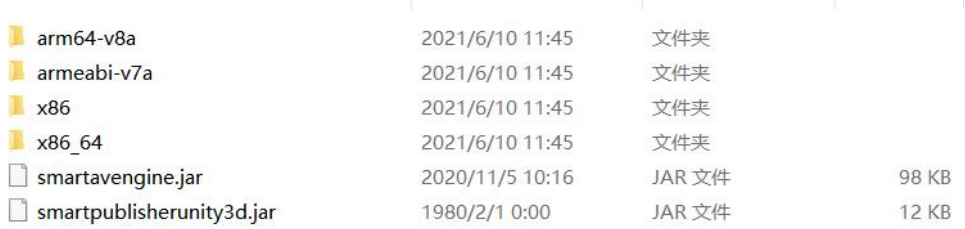
4. Android 推送端 SDK 说明

4.1 demo 说明

SmartU3dAndroidPublisher：大牛直播 SDK Unity3D Android 平台 RTMP 直播推送 SDK 工程 demo，以采集 Unity 窗体和 Unity 的音频数据为例。

4.2 集成说明

- Unity3D 接口和调用 demo，参见：SmartPublisherAndroidMono.cs
- SmartU3dPublisher\Assets\Plugins\Android\libs 下相关库到工程：



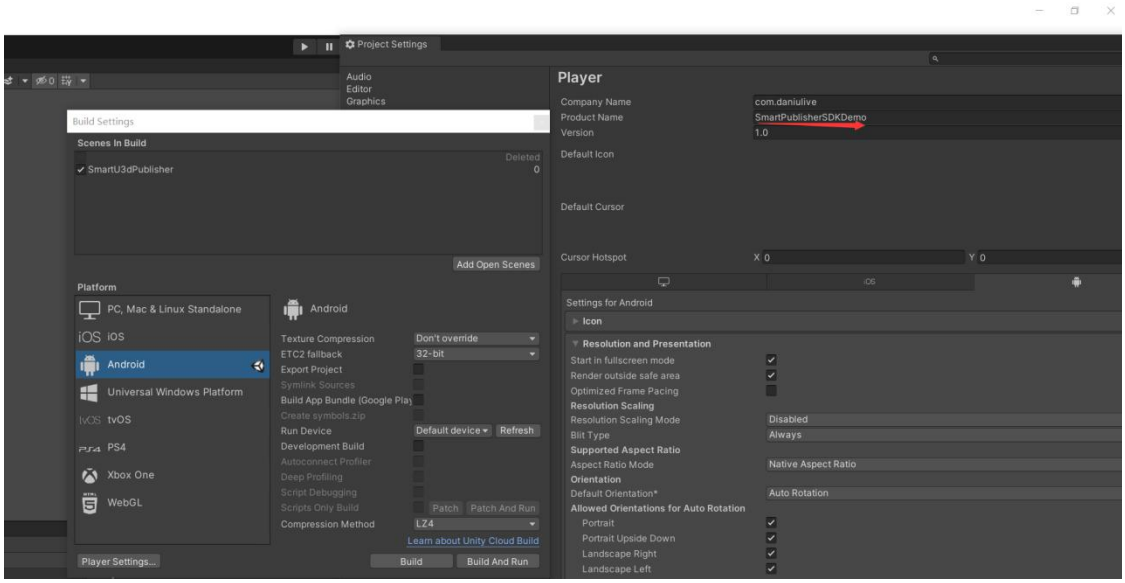
- smartavengine.jar 加入到工程；
- smartpulisherunity3d.jar 加入工程；
- 32/64 位 arm 和 x86 下 libSmartPublisher.so；
- 在 SmartU3dPublisher\Assets\Plugins\Android\AndroidManifest.xml 配置相关权限：

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" ></uses-permission>
<uses-permission android:name="android.permission.INTERNET" ></uses-permission>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
```

- 如需集成到自己系统测试，请用大牛直播 SDK 的 app name(不然集成提示 license failed)，正式授权版按照授权 app name 正常使用即可；
- 正式授权版，需要调用 license 设置接口，确保 license key 正确设置方可正常使用；
- 如何改 app-name：

  Unity3D 模式下：File-->Build Settings-->Android-->Player Settings:



4.3 调用时序

1. **【最先调用】**NT_PB_U3D_Init，推送实例初始化，目前预留；

2. **【正式授权版设置 license key】**NT_PB_U3D_SetSDKClientKey，设置正式授权版 license，请早于 NT_PB_U3D_Open 接口调用；

3. **【设置音频采集类型】**NT_PB_U3D_SetAudioCaptureType，如不设置，默认采集内置麦克风音频数据，请确保在 NT_U3D_Open()之前调用！采集类型 0: 默认内置麦克风采集; 1: 外部 PCM 采集 如需采集 Unity 的音频数据，请选择 1。

4. **【获得推送实例句柄】**NT_PB_U3D_Open，设置上下文信息，返回推送实例句柄；

5. 【设置 GameObject】NT_PB_U3D_Set_Game_Object，注册 Game Object，用于消息传递；

6. 【设置 H.264 硬编码】NT_PB_U3D_SetVideoHWEncoder，设置特定机型 H.264 硬编码；

7. 【设置 H.265 硬编码】NT_PB_U3D_SetVideoHevcHWEncoder，设置特定机型 H.265 硬编码；

8. 【设置软编码可变码率编码】NT_PB_U3D_SetSwVBRMode，设置软编码可变码率编码模式；

9. 【设置关键帧间隔】NT_PB_U3D_SetGopInterval，设置关键帧间隔，一般来说，关键帧间隔可以设置到帧率的 2-4 倍；

10. 【软编码编码码率】NT_PB_U3D_SetSWVideoBitRate，设置软编码编码码率；

11. 【设置帧率】NT_PB_U3D_SetFPS，设置视频帧率；

12. 【设置编码 profile】NT_PB_U3D_SetSWVideoEncoderProfile，设置软编码编码 profile；

13. 【设置软编码编码速度】NT_PB_U3D_SetSWVideoEncoderSpeed，设置软编码编码速度；

14. 【实时静音】NT_PB_U3D_SetMute，设置推送过程中，音频实时静音；

15. 【输入音量调节】NT_PB_U3D_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；

16. 【设置音频编码类型】NT_PB_U3D_SetAudioCodecType，设置音频编码类型，默认 AAC 编码；

17. 【设置音频编码码率】NT_PB_U3D_SetAudioBitRate，设置音频编码码率，当前只对 AAC 编码有效；

18. 【设置 speex 音频编码质量】NT_PB_U3D_SetSpeexEncoderQuality，设置 speex 音频编码质量；

19. 【设置音频噪音抑制】NT_PB_U3D_SetNoiseSuppression，设置音频噪音抑制；

20. 【设置音频自动增益控制】NT_PB_U3D_SetAGC，设置音频自动增益控制；

21. 【设置快照 flag】NT_PB_U3D_SetSaveImageFlag，设置是否需要在推流或录像过程中快照；

22. 【实时快照】NT_PB_U3D_SaveCurImage，推流或录像过程中快照；

23. 【录像音频控制】NT_PB_U3D_SetRecorderAudio，音频录制开关，目的是为了更细粒度的去控制录像，一般不需要调用这个接口，这个接口使用场景比如同时推送音视频，但只想录制视频，可以调用这个接口关闭音频录制；

24. 【录像视频控制】NT_PB_U3D_SetRecorderVideo，视频录制开关，目的是为了更细粒度的去控制录像，一般不需要调用这个接口，这个接口使用场景比如同时推送音视频，但只想录制音频，可以调用这个接口关闭视频录制；

25. 【录像】NT_PB_U3D_CreateFileDirectory，创建录像存储路径；

26. 【录像】NT_PB_U3D_SetRecorderDirectory，设置录像存储路径；

27. 【录像】NT_PB_U3D_SetRecorderFileMaxSize，设置单个录像文件大小；

28. 【数据投递】NT_PB_U3D_OnCaptureVideoI420Data，实时投递 YUV 数据；

29. 【数据投递】NT_PB_U3D_OnCaptureVideoRGB24Data，实时投递 RGB24 数据；

30. 【数据投递】NT_PB_U3D_OnCaptureVideoRGBA32Data，实时投递 RGBA 数据；

31. 【数据投递】NT_PB_U3D_OnPCMData，实时投递 PCM 数据，数据类型，byte 数组；

32. 【数据投递】NT_PB_U3D_OnPCMShortArray，实时投递 PCM 数据，数据类型 short 数组；

33. 【推送 URL】NT_PB_U3D_SetPushUrl，设置推送的 RTMP URL；

34. 【开始 RTMP 推流】NT_PB_U3D_StartPublisher，开始 RTMP 推流；

35. 【停止 RTMP 推流】NT_PB_U3D_StopPublisher，停止 RTMP 推流；

36. 【开始录像】NT_PB_U3D_StartRecorder，开始录像；

37. 【暂停录像】NT_PB_U3D_PauseRecorder，暂停录像；

38. 【停止录像】NT_PB_U3D_StopRecorder，停止录像；

39. **【关闭】**NT_U3D_Close，关闭推送实例；

40. **【最后调用】**NT_PB_U3D_UnInit，UnInit 推送 SDK，最后调用。

4.4 相关实现

Android 平台 Unity3D 调用的是大牛直播 SDK 的原生接口封装，接口用法基本和原生接口匹配。

1. 基础初始化

```
private void Start()
{
    game_object_ = this.gameObject.name;      //获取 GameObject Name

    AndroidJavaClass android_class = new AndroidJavaClass("com.unity3d.player.UnityPlayer");
    java_obj_cur_activity_ = android_class.GetStatic<AndroidJavaObject>("currentActivity");
    pusher_obj_ = new AndroidJavaObject("com.daniulive.smartpublisher.SmartPublisherUnity3d");

    NT_PB_U3D_Init();

    //NT_U3D_SetSDKClientKey("", "", 0);

    audioOptionSel.onValueChanged.AddListener(SelAudioPushType);

    btn_encode_mode_.onClick.AddListener(OnEncodeModeBtnClicked);
```

```
        btn_pusher_.onClick.AddListener(OnPusherBtnClicked);

        btn_mute_.onClick.AddListener(OnMuteBtnClicked);
    }
```

2．调用 Open()接口，获取推送实例

```
    public void Push()
    {
        if (is_running_)
        {
            Debug.Log("已推送..");
            return;
        }

        video_width_ = Screen.width;
        video_height_ = Screen.height;

        //获取输入框的 url
        push_url_ = input_url_.text.Trim();

        OpenPusher();

        if (pusher_handle_ == 0)
            return;

        NT_PB_U3D_Set_Game_Object(pusher_handle_, game_object_);

        /* ++ 推送前参数配置可加在此处 ++ */

        InitAndSetConfig();

        NT_PB_U3D_SetPushUrl(pusher_handle_, push_url_);
        /* -- 推送前参数配置可加在此处 -- */

        int flag = NT_PB_U3D_StartPublisher(pusher_handle_);

        if (flag  == DANIULIVE_RETURN_OK)
        {
            Debug.Log("推送成功..");
        }
        else
        {
            Debug.LogError("推送失败..");
        }

        is_running_ = true;

        if (audio_push_type_ == 1)
        {
            PostUnityAudioClipData();
        }
    }
```

3.OpenPusher 实现

OpenPusher 主要是调用底层 SDK 的 Open 接口，创建推送实例，并返回推送句柄。

```
    private void OpenPusher()
    {
        if ( java_obj_cur_activity_ == null )
        {
            Debug.LogError("getApplicationContext is null");
            return;
        }
```

```
        NT_PB_U3D_SetAudioCaptureType(audio_push_type_);

        int audio_opt = 1;
        int video_opt = 1;

        pusher_handle_ = NT_PB_U3D_Open(audio_opt, video_opt, video_width_, video_height_);

        if (pusher_handle_ != 0)
            Debug.Log("NT_PB_U3D_Open success");
        else
            Debug.LogError("NT_PB_U3D_Open fail");
    }
```

4. InitAndSetConfig 具体实现

InitAndSetConfig 主要完成 SDK 的一些参数设定工作，如软硬编码设定、码率设定、音频采集类型、视频帧率、码率设置等。

```
    private void InitAndSetConfig()
    {
        if (is_hw_encode_)
        {
            int h264HWKbps = setHardwareEncoderKbps(true, video_width_, video_height_);

            Debug.Log("h264HWKbps: " + h264HWKbps);

            int isSupportH264HWEncoder = NT_PB_U3D_SetVideoHWEncoder(pusher_handle_, h264HWKbps);

            if (isSupportH264HWEncoder == 0)
            {
                Debug.Log("Great, it supports h.264 hardware encoder!");
            }
        }
        else {
            if (is_sw_vbr_mode_) //H.264 software encoder
            {
                int is_enable_vbr = 1;
                int video_quality = CalVideoQuality(video_width_, video_height_, true);
                int vbr_max_bitrate = CalVbrMaxKBitRate(video_width_, video_height_);

                NT_PB_U3D_SetSwVBRMode(pusher_handle_, is_enable_vbr, video_quality, vbr_max_bitrate);
                //NT_PB_U3D_SetSWVideoEncoderSpeed(pusher_handle_, 2);
            }
        }

        NT_PB_U3D_SetAudioCodecType(pusher_handle_, 1);

        NT_PB_U3D_SetFPS(pusher_handle_, 25);

        NT_PB_U3D_SetGopInterval(pusher_handle_, 25*2);

        //NT_PB_U3D_SetSWVideoBitRate(pusher_handle_, 600, 1200);
    }
```

5. 停止推送

```
    private void ClosePusher()
    {
        if (audio_push_type_ == 1)
        {
            StopAudioSource();
        }
```

```
        if (texture_ != null)
        {
            UnityEngine.Object.Destroy(texture_);
            texture_ = null;
        }

        int flag = NT_PB_U3D_StopPublisher(pusher_handle_);

        if (flag == DANIULIVE_RETURN_OK)
        {
            Debug.Log("停止成功..");
        }
        else
        {
            Debug.LogError("停止失败..");
        }

        flag = NT_PB_U3D_Close(pusher_handle_);

        if (flag == DANIULIVE_RETURN_OK)
        {
            Debug.Log("关闭成功..");
        }
        else
        {
            Debug.LogError("关闭失败..");
        }

        pusher_handle_ = 0;

        NT_PB_U3D_UnInit();

        is_running_ = false;
    }
```

6．数据采集

摄像头和屏幕的数据采集，还是调用原生的 SDK 接口，本文不再赘述，如果需要采集 Unity 窗体的数据，可以用参考以下代码：

```
        if (texture_ == null || video_width_ != Screen.width || video_height_ != Screen.height)
        {
            Debug.Log("OnPostRender screen changed++ scr_width: " + Screen.width + " scr_height: " + Screen.height);

            if (texture_ !=  null)
            {
                UnityEngine.Object.Destroy(texture_);
                texture_ = null;
            }

            video_width_ = Screen.width;
            video_height_ = Screen.height;

            scale_width_ = (video_width_ + 1) / 2;
            scale_height_ = (video_height_ + 1) / 2;

            if (scale_width_ % 2 != 0)
            {
                scale_width_ = scale_width_ + 1;
            }

            if (scale_height_ % 2 != 0)
```

```
            {
                scale_height_ = scale_height_ + 1;
            }

            //texture_ = new Texture2D(video_width_, video_height_, TextureFormat.RGB24, false);
            texture_ = new Texture2D(video_width_, video_height_, TextureFormat.RGBA32, false);

            Debug.Log("OnPostRender screen changed--");

            return;
        }

        texture_.ReadPixels(new Rect(0, 0, video_width_, video_height_), 0, 0, false);
```

从 texture 里面，通过调用 GetRawTextureData()，获取到原始数据。

如需采集 Unity 的 AudioClip 数据：

```
/// <summary>
///  获取 AudioClip 数据
/// </summary>
private void PostUnityAudioClipData()
{
    TimerManager.UnRegister(this);

    audio_clip_info_ = new AudioClipInfo();

    audio_clip_info_.audio_source_ = GameObject.Find("Canvas/Panel/AudioSource").GetComponent<AudioSource>();

    if (audio_clip_info_.audio_source_ == null)
    {
        Debug.LogError("audio source is null..");
        return;
    }

    if (audio_clip_info_.audio_source_.clip == null)
    {
        Debug.LogError("audio clip is null..");
        return;
    }

    audio_clip_info_.audio_clip_ = audio_clip_info_.audio_source_.clip;
    audio_clip_info_.audio_clip_offset_ = 0;
    audio_clip_info_.audio_clip_length_ = audio_clip_info_.audio_clip_.samples * audio_clip_info_.audio_clip_.channels;

    pcm_mute_data_ = FillPcmMuteData(audio_clip_info_.audio_clip_);

    TimerManager.Register(this, 0.019999f, (PostPCMData), false);
}
```

7. 数据对接

Unity 的视频数据，获取到 Texture 数据后，通过调用 NT_PB_U3D_OnCaptureVideoRGBA32Data()接口，传递给 SDK 层。

如果是 Unity 的 AudioClip 采集的数据，调用 NT_PB_U3D_OnPCMShortArray()传递给 SDK。

```
NT_PB_U3D_OnPCMShortArray(pusher_handle_, pcm_data.data,
    0, pcm_data.size_,
    pcm_data.sample_rate_,
    pcm_data.channels_,
    pcm_data.per_channel_sample_number_);
```

8．相关 event 回调处理

```
/// <summary>
/// android 传递过来 code
/// </summary>
/// <param name="code"></param>
public void onNTSmartEvent(string param)
{
    if (!param.Contains(","))
    {
        Debug.Log("[onNTSmartEvent] android 传递参数错误");
        return;
    }

    string[] strs = param.Split(',');

    string player_handle =strs[0];
    string code = strs[1];
    string param1 = strs[2];
    string param2 = strs[3];
    string param3 = strs[4];
    string param4 = strs[5];

    Debug.Log("[onNTSmartEvent] code: 0x" + Convert.ToString(Convert.ToInt32(code), 16));

     String publisher_event = "";

     switch (Convert.ToInt32(code))
     {
        case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_STARTED:
            publisher_event = "开始..";
            break;
        case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_CONNECTING:
            publisher_event = "连接中..";
            break;
        case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_CONNECTION_FAILED:
            publisher_event = "连接失败..";
            break;
        case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_CONNECTED:
            publisher_event = "连接成功..";
            break;
        case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_DISCONNECTED:
            publisher_event = "连接断开..";
            break;
        case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_STOP:
            publisher_event = "关闭..";
            break;
        case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_RECORDER_START_NEW_FILE:
            publisher_event = "开始一个新的录像文件 :" + param3;
            break;
        case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_ONE_RECORDER_FILE_FINISHED:
            publisher_event = "已生成一个录像文件 :" + param3;
            break;

        case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_SEND_DELAY:
            publisher_event = "发送时延: " + param1 + " 帧数:" + param2;
            break;

        case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_CAPTURE_IMAGE:
            publisher_event = "快照: " + param1 + " 路径：" + param3;

            if (Convert.ToInt32(param1) == 0)
            {
                publisher_event = publisher_event + "截取快照成功..";
```

```
                }
                else
                {
                    publisher_event = publisher_event + "截取快照失败..";
                }
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PUBLISHER_RTSP_URL:
                publisher_event = "RTSP 服务 URL: " + param3;
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PUSH_RTSP_SERVER_RESPONSE_STATUS_CODE:
                publisher_event = "RTSP status code received, codeID: " + param1 + ", RTSP URL: " + param3;
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PUSH_RTSP_SERVER_NOT_SUPPORT:
                publisher_event = "服务器不支持 RTSP 推送，推送的 RTSP URL: " + param3;
                break;
        }

        Debug.Log(publisher_event);

    }
```

5. iOS 播放端 SDK 说明

5.1 demo 说明

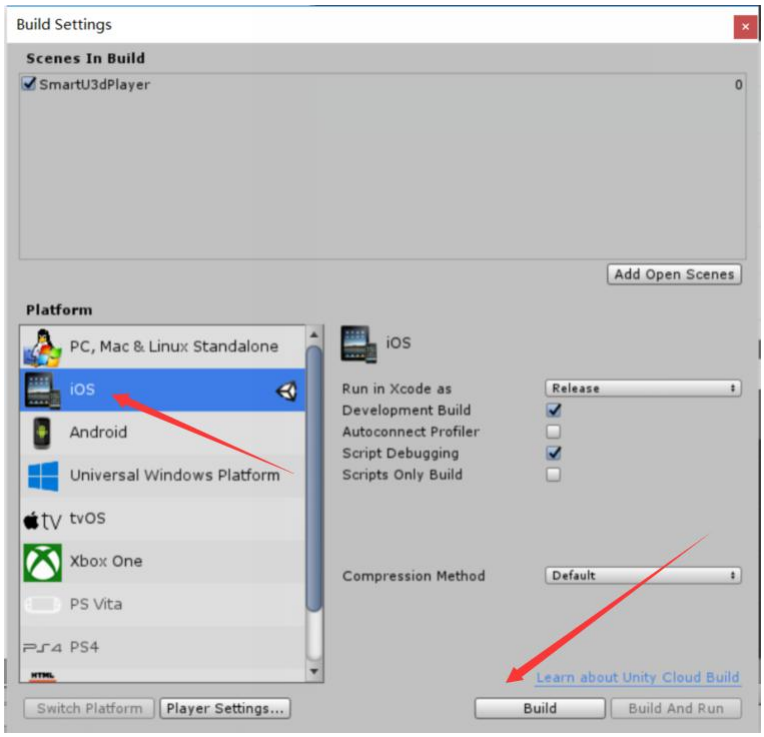
- SmartU3diOSPlayer：大牛直播 SDK Unity3D iOS 平台 RTMP/RTSP 直播播放端工程。

5.2 集成说明

拷贝以下文件，到 Assets-->Plugins-->iOS 目录：

相关头文件和调用说明，参见：SmartPlayeriOSMono.cs

Unity3D 工程下，File-->Build Settings，Platform 选择 iOS，然后点击 build，设置目录，生成 xcode 工程：



生成后的 xcode 工程，添加以下依赖库：

- 相关库：libSmartPlayerSDK.a
- 引入以下依赖 framework
  - libbz.tbd
  - Libbz2.tbd
  - libiconv.tbd
  - libstdc++.tbd
  - Libc++.tbd
  - Accelerate.framework
  - AssetsLibrary.framework
  - AudioToolBox.framework
  - AVFoundation.framework
  - CoreMedia.framework
  - Foundation.framework
  - GLKit.framework
  - OpenGLES.framework
  - UIKit.framework
  - VideoToolBox.framework
- 如需集成到自己系统测试，请用大牛直播的 app name：

  Info.plist-->右键 Open As-->Source Code

  添加或者编辑

  <key>CFBundleName</key>

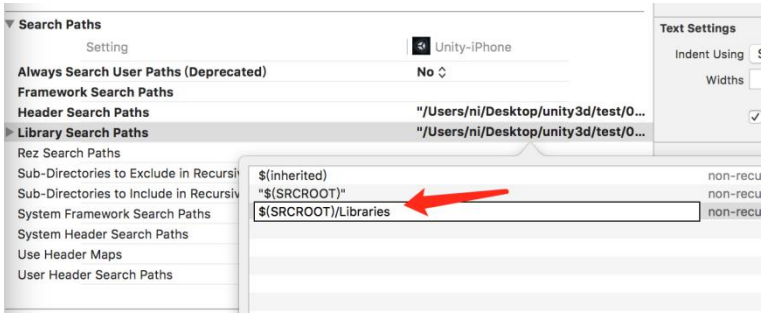
  <string>SmartiOSPlayer</string>
- 快照添加到“照片”权限：

  Info.plist-->右键 Open As-->Source Code

  添加

  <key>NSPhotoLibraryUsageDescription</key>

  <string>1</string>
- 导出后的 xcode 工程，如编译不过，参考以下设置：



5.3 调用时序

1. 【最先调用】NT_U3D_Init：player 初始化，目前预留；

2. **【正式授权版设置 license key】**NT_U3D_SetSDKClientKey，设置正式授权版 license，请早于 NT_U3D_Open 接口调用；

3. **【获得 player 句柄】**NT_U3D_Open，设置上下文信息，返回 player 句柄；

4. 【设置 GameObject】NT_U3D_Set_Game_Object，注册 Game Object，用于消息传递；

5. 【设置硬解码】NT_U3D_SetVideoDecoderMode，设置是否用硬解码播放，如硬解码不支持，切换到软解码；

6. 【缓冲设置】NT_U3D_SetBuffer，设置播放端缓存数据 buffer，以毫秒(ms)为单位，如超低延迟模式下，不需 buffer 数据，设置为 0；

7. 【RTSP TCP/UDP 设置】NT_U3D_SetRTSPTcpMode，设置 TCP/UDP 播放模式，**注意：此接口仅用于 RTSP；**

8. 【RTSP 超时时间设置】NT_U3D_SetRTSPTimeout，设置 RTSP 超时时间, timeout 单位为秒，必须大于 0；

9. 【RTSP TCP/UDP 自动切换】NT_U3D_SetRTSPAutoSwitchTcpUdp，设置 RTSP TCP/UDP 自动切换，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；

10. **【实时静音-可实时调用】**NT_U3D_SetMute，设置播放过程中，实时静音/取消静音；

11. 【快速启动】NT_U3D_SetFastStartup，Set fast startup(快速启动)，设置快速启动后，如果 CDN 缓存 GOP，daniulive player 可快速出帧；

12. 【低延迟模式】NT_U3D_SetPlayerLowLatencyMode，设置低延迟模式；

13. **【视频垂直反转-可实时调用】**NT_U3D_SetFlipVertical，**视频垂直反转；**

14. **【视频水平反转-可实时调用】**NT_U3D_SetFlipHorizontal，**视频水平反转；**

15. **【视频显示角度设置-可实时调用】**NT_U3D_SetRotation，**针对类似于安防摄像头或其他设备出来的图像倒置现象，支持视频播放 view 顺时针旋转，当前支持 0 度，90 度, 180 度, 270 度 旋转，注意除了 0 度之外， 其他角度都会额外消耗性能；**

16. 【下载速度回调设置】NT_U3D_SetReportDownloadSpeed，设置下载速度上报，默认不上报下载速度；

17. 【快照设置】NT_U3D_SetSaveImageFlag，设置是否需要在播放或录像过程中快照；

18. **【快照-录像或播放后，可随时调用】**NT_U3D_SaveCurImage，播放过程中，根据设置路径和文件名，实时快照；

19. **【快速切换 url-可实时调用】**NT_U3D_SwitchPlaybackUrl，快速切换播放 url，快速切换时，只换播放 source 部分，适用于不同数据流之间，快速切换；

20. 【录像设置】NT_U3D_CreateFileDirectory，创建文件路径，注意：iOS 只提供接口，未提供具体实现；

21. 【录像设置】NT_U3D_SetRecorderDirectory，设置文件路径；

22. 【录像设置】NT_U3D_SetRecorderFileMaxSize，设置每个录像文件最大 size，以兆（M）为单位，范围[5M~500M]；

23. 【录像设置】NT_U3D_SetRecorderAudioTranscodeAAC，支持拉取的 RTMP/RTSP 的 PCMA/PCMU/SPEEX 音频格式转 AAC 后录制；

24. 【设置播放或录像 URL】NT_U3D_SetUrl，设置播放/录像 url；

25. **【播放】**NT_U3D_StartPlay，开始播放；

26. 【播放】NT_U3D_GetVideoFrame，获取底层回调的 YUV 数据；

27. 【播放】NT_U3D_StopPlay，停止播放；

28. 【录像】NT_U3D_StartRecorder，开始录像；

29. 【录像】NT_U3D_StopRecorder，停止录像；

30. 【关闭】NT_U3D_Close，关闭播放器实例；

31. 【最后调用】NT_U3D_UnInit，UnInit Player，最后调用。

5.4 相关实现

1. 初始化播放

完成相关的初始化和接口参数设定后，调用 NT_U3D_StartPlay()实现音视频数据播放。

```
public void Play(int sel)
{
    if (videoctrl[sel].is_running)
    {
        Debug.Log("已经在播放。。");
        return;
    }

    videoctrl[sel].player_handle_ = NT_U3D_Open();

    if (videoctrl[sel].player_handle_ == IntPtr.Zero)
    {
        Debug.LogError("open fail");
        return;
    }

    NT_U3D_Set_Game_Object(videoctrl[sel].player_handle_, game_object_);

    /* ++ 播放前参数配置可加在此处 ++ */
    int is_using_tcp = 1;            //TCP 模式
    NT_U3D_SetRTSPTcpMode(videoctrl[sel].player_handle_, is_using_tcp);

    int is_report = 0;
    int report_interval = 1;
    NT_U3D_SetReportDownloadSpeed(videoctrl[sel].player_handle_, is_report, report_interval);   //下载速度回调

    int play_buffer_time_ = 100;
    NT_U3D_SetBuffer(videoctrl[sel].player_handle_, play_buffer_time_);                         //设置 buffer time

    int is_low_latency_ = 0;
    NT_U3D_SetPlayerLowLatencyMode(videoctrl[sel].player_handle_, is_low_latency_);     //设置是否启用低延迟模式

    int is_mute_ = 0;
    NT_U3D_SetMute(videoctrl[sel].player_handle_, is_mute_);                            //是否启动播放的时候静音
```

```
    int is_hw_decode_ = 1;
    NT_U3D_SetVideoDecoderMode(videoctrl[sel].player_handle_, is_hw_decode_);                    //设置软硬解模式

    int is_fast_startup = 1;
    NT_U3D_SetFastStartup(videoctrl[sel].player_handle_, is_fast_startup);                       //设置快速启动模式

    int rtsp_timeout = 10;
    NT_U3D_SetRTSPTimeout(videoctrl[sel].player_handle_, rtsp_timeout);                          //设置 RTSP 超时时间

    int is_auto_switch_tcp_udp = 1;
    NT_U3D_SetRTSPAutoSwitchTcpUdp(videoctrl[sel].player_handle_, is_auto_switch_tcp_udp);       //设置 TCP/UDP 模式自动
切换

    NT_U3D_SetUrl(videoctrl[sel].player_handle_, videoctrl[sel].videoUrl);
    /* -- 播放前参数配置可加在此处 -- */

    int isEnableYuvBlock = 1;   //是否回调 YUV 数据
    int flag = NT_U3D_StartPlay(videoctrl[sel].player_handle_, isEnableYuvBlock);

    if (flag == DANIULIVE_RETURN_OK)
    {
        videoctrl[sel].is_need_init_texture_ = true;
        videoctrl[sel].is_need_get_frame_ = true;
        Debug.Log("播放成功");
    }
    else
    {
        videoctrl[sel].is_need_get_frame_ = false;
        Debug.Log("播放失败");
    }

    videoctrl[sel].is_running = true;
}
```

2. 停止播放：

调用 NT_U3D_StopPlay()后，如果没有录像，调用 NT_U3D_Close()，播放 handler 置空，然后调用 NT_U3D_UnInit()即可。

```
private void ClosePlayer(int sel)
{
    videoctrl[sel].is_need_get_frame_ = false;
    videoctrl[sel].is_need_init_texture_ = false;

    int flag = NT_U3D_StopPlay(videoctrl[sel].player_handle_);

    if (flag == DANIULIVE_RETURN_OK)
    {
        Debug.Log("停止成功");
    }
    else
    {
        Debug.Log("停止失败");
    }

    flag = NT_U3D_Close(videoctrl[sel].player_handle_);
    if (flag == DANIULIVE_RETURN_OK)
    {
        Debug.Log("关闭成功");
    }
    else
    {
        Debug.Log("关闭失败");
    }
```

```
        videoctrl[sel].player_handle_ = IntPtr.Zero;

        videoctrl[sel].is_running = false;
    }
```

3. 相关 Event 处理：

```
    /// <summary>
    /// iOS 传递过来 code
    /// </summary>
    /// <param name="code"></param>
    public void onNTSmartEvent(string param)
    {
        if (!param.Contains(","))
        {
            Debug.Log("[onNTSmartEvent] iOS 传递参数错误");
            return;
        }

        string[] strs = param.Split(',');

        string player_handle = strs[0];
        string code = strs[1];
        string param1 = strs[2];
        string param2 = strs[3];
        string param3 = strs[4];
        string param4 = strs[5];

        Debug.Log("[onNTSmartEvent] code: 0x" + Convert.ToString(Convert.ToInt32(code), 16));

        switch (Convert.ToInt32(code))
        {
            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_STARTED:
                Debug.Log("开始。。");
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_CONNECTING:
                Debug.Log("连接中。。");
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_CONNECTION_FAILED:
                Debug.Log("连接失败。。");
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_CONNECTED:
                Debug.Log("连接成功。。");
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_DISCONNECTED:
                Debug.Log("连接断开。。");
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_STOP:
                Debug.Log("停止播放。。");
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_RESOLUTION_INFO:
                Debug.Log("分辨率信息: width: " + Convert.ToInt32(param1) + ", height: " + Convert.ToInt32(param2));
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_NO_MEDIADATA_RECEIVED:
                Debug.Log("收不到媒体数据，可能是 url 错误。。");
                break;
            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_SWITCH_URL:
                Debug.Log("切换播放 URL。。");
                break;

            case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_CAPTURE_IMAGE:
                Debug.Log("快照: " + param1 + " 路径：" + param3);

                if (Convert.ToInt32(param1) == 0)
```

```
            {
                Debug.Log("截取快照成功。.");
            }
            else
            {
                Debug.LogError("截取快照失败。.");
            }
            break;

        case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_RECORDER_START_NEW_FILE:
            Debug.Log("[record]开始一个新的录像文件：" + param3);
            break;
        case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_ONE_RECORDER_FILE_FINISHED:
            Debug.Log("[record]已生成一个录像文件：" + param3);
            break;

        case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_START_BUFFERING:
            Debug.Log("Start_Buffering");
            break;

        case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_BUFFERING:
            Debug.Log("Buffering: " + Convert.ToInt32(param1));
            break;

        case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_STOP_BUFFERING:
            Debug.Log("Stop_Buffering");
            break;

        case EVENTID.EVENT_DANIULIVE_ERC_PLAYER_DOWNLOAD_SPEED:
            Debug.Log("download_speed:" + param1 + "Byte/s" + ", "
            + (Convert.ToInt32(param1) * 8 / 1000) + "kbps" + ", " + (Convert.ToInt32(param1) / 1024)
            + "KB/s");
            break;
    }
}
```