SDK 可用性检测方案

1 SDK 接入注意事项（必看）

> **请开发者注意：** SDK 的每个回调都在一个独立且唯一工作线程中抛出（SDK 主线程），开发者接收到回调之后务必需要切换线程，不阻塞 SDK 主线程。否则，可能会发生 SDK 主线程卡死等异常问题。

2 检测设备可用性

需求场景： 当开发者想要在推流前先检测设备是否可用，以提示用户当前设备的运行情况，则可用到如下的 “设备可用性检测方案”。

2.1 检测麦克风（输入）设备是否可用

开发者通过 `SoundLevel` 模块中的 `OnCaptureSoundLevelUpdate` 回调，可以在自己的应用程序中以音浪UI的形式向用户展示当前麦克风的音浪变化，以检测当前麦克风是否可用。

示例代码如下：

- 1. 继承 `IZegoSoundLevelCallback` 类并且实现里面的两个回调函数。

```
#include "zego-api-sound-level.h"

class MySoundLevelCallback : public IZegoSoundLevelCallback
{
    public:
        ......

        /**
        获取 soundLevel 更新的回调

        @param vecSoundLevels 房间内所有流（非自己推的流）的 soundLevel 值
        */
        void OnSoundLevelUpdate(ZegoSoundLevelInfo *pSoundLevelList, unsigned int sou
ndLevelCount) override
        {
```

```
            //切换线程
            ......

            //保存拉流的soundlevel列表，可以在UI上实时展示每条流音浪的变化情况
            ......
        }

        /**
         获取 captureSoundLevel 更新的回调

         @param pCaptureSoundLevel 房间内采集 soundLevel 值（自己推的流）
         */
        void OnCaptureSoundLevelUpdate(ZegoSoundLevelInfo *pCaptureSoundLevel) overri
de
        {
            //切换线程
            ......

            //可以在UI上实时展示自己音浪的变化情况，以检测麦克风设备是否可用
            ......
        }
};
```

- 2. 设置 `IZegoSoundLevelCallback` 类的回调监听

```
#include "zego-api-sound-level.h"


  //创建 SoundLevel 回调对象
  MySoundLevelCallback *callback = new MySoundLevelCallback;
  //设置获取 soundLevel 的回调对象
  SOUNDLEVEL::SetSoundLevelCallback(callback);
```

- 3. 分别通过 `SetSoundLevelMonitorCycle`、`StartSoundLevelMonitor` 和 `StopSoundLevelMonitor` 进行音量监听的间隔时间的设置、开始监听音量以及停止监听音量。

```
#include "zego-api-sound-level.h"

//设置 soundLevel 的监控周期
SOUNDLEVEL::SetSoundLevelMonitorCycle(200);

//启动 soundLevel 监听
SOUNDLEVEL::StartSoundLevelMonitor();
```

```
//停止 soundLevel 监听
SOUNDLEVEL::StopSoundLevelMonitor();
```

2.2 检测扬声器是否可用

使用 `MediaPlayer` 模块进行一小段音频的播放，以让用户能够检测扬声器是否可用。 **开发者只需要听音频文件是否播放正常即可判断当前扬声器是否可用。**

示例代码如下：

- 1. 获取播放器

```
#include "zego-api-mediaplayer.h"

//获取播放器，播放器只有在引擎启动的情况下才有效，需要在 InitSDK 后再调用，且不能和 InitSDK 在同一个函数里调用。
 AVE::IMediaPlayer* myPlayer = GetMediaPlayer(AVE::IMediaPlayer::PlayerType::Type_AUX);
```

- 2. 调用播放 引擎启动后获取到 `MediaPlayer` 对象，调用对象的 `void Start(const char* path, bool repeat_play = false)` 函数进行音频播放。其中，`path` 为将要播放的文件路径，`repeat_play` 表示标识是否需要重复播放。

```
#include "zego-api-mediaplayer.h"

//启动播放器
myPlayer->Start("D:¥¥test.mp3", true);
```

注意： 目前MediaPlayer仅支持mp3/mp4/m4a/mkv格式的媒体文件。

2.3 检测摄像头设备是否可用

- 若开发者决定使用 SDK 内部渲染，则需要调用如下接口将需要渲染的窗口句柄传入 SDK，让 SDK 对该窗口进行渲染（以推流为例）。开发者只需要观察窗口的渲染是否正常即可判断当前摄像头是否可用。

```
#include "LiveRoom-Publisher.h"

 //设置本地预览视图，第一个参数传入需要渲染的窗口句柄
 LIVEROOM::SetPreviewView(pView->winId(), AV::PUBLISH_CHN_MAIN);
```

- 若开发者决定使用外部采集&外部渲染的方式，则从摄像头抛出数据的回调中将视频帧数据自行渲染到所需的窗体上。开发者同样只需观察窗口的渲染是否正常即可判断当前摄像头是否可用。

```
#include "video_capture.h"

class MyExternalClient : public AVE::VideoCaptureDevice::Client
{
public:
    //......

    //开启外部采集后，Camera对象回调采集的视频帧
    void OnIncomingCapturedData(
        const char* data,
        int length,
        const AVE::VideoCaptureFormat& frame_format,
        unsigned long long reference_time,
        unsigned int reference_time_scale) override
    {
        //通过该接口回调出的视频帧，开发者自行渲染到窗口中
    }

};
```

3 检测设备状态

需求场景： 当设备已正常运行，开发者想要对设备的运行状态作检测，以向用户实时抛出设备状态信息，则可用到如下的 “设备状态检测方案”。

示例代码如下：

- 1）继承 `IZegoDeviceStateCallback` 类并根据需要实现其中的监听回调函数

```
#include "AVDefine.h"

class MyDeviceStateCallback : public IZegoDeviceStateCallback
{
public:
    //......

    //当音频设备音量被改变（如开发者自行调用音量设置的接口）时，SDK 会回调该函数。
    void OnAudioVolumeChanged(AudioDeviceType deviceType, const char *deviceId, Volume
Type volumeType, unsigned int volume, bool bMuted)
    {
        //切换线程
        ......

        //开发者可以在抛出该回调时向用户展示UI。如：展示当前音频设备的实时音量值等。
        ......
    }

    //当音频设备插入或者拔出时，SDK会回调该函数。
    void OnAudioDeviceStateChanged(AudioDeviceType deviceType, DeviceInfo *deviceInfo,
DeviceState state)
    {
        //切换线程
        ......

        //开发者可以在抛出该回调时向用户展示UI。如：展示当前音频设备的实时列表、动态切
换音频设备等。
        ......
    }

    //当视频设备插入或者拔出时，SDK会回调该函数。
    void OnVideoDeviceStateChanged(DeviceInfo *deviceInfo, DeviceState state)
    {
        //切换线程
        ......

        //开发者可以在抛出该回调时向用户展示UI。如：展示当前视频设备的实时列表、动态切
换视频设备等。
        ......
    }

    //当视频设备显示异常时（如多个程序抢占同一个摄像头导致画面不显示），SDK会回调该函
数。
    void OnDeviceError(const char* deviceName, int errorCode)
    {
        //切换线程
        ......

        //开发者可以在抛出该回调时向用户展示UI。如：展示当前摄像头异常的弹窗，并要求用
户自行检查或者重新插拔的提示语等。
        ......
```

```
    }
}
```

- 2. 设置 `IZegoDeviceStateCallback` 类的回调监听

```
#include "LiveRoom.h"

//创建设备状态回调对象
MyDeviceStateCallback *callback = new MyDeviceStateCallback;
//设置音频视频设备变化的回调
LIVEROOM::SetDeviceStateCallback(callback);
```