实时互动-工业能源版

远端设备 SDK

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

远端设备 SDK
基本介绍

最近更新时间：2023-05-12 18:00:03

适用范围

本 SDK 用于远端设备接入, 适用于 Linux 和 Windows 64 位系统，提供 so、dll 动态链接库形式接入。

主要功能

本 SDK 面向远程实时操控场景提供低时延音视频通话和控制/状态数据传输，主要功能如下：

视频渲染

在指定句柄的窗口进行视频渲染，并能够自动响应窗口的缩放事件。

视频解码

支持 H264、H265 和 AV1 等视频格式解码。

视频传输

支持 H264、H265 和 AV1 等编码视频流的传输，具备抗弱网和低延迟传输能力。

二进制数据传输

支持二进制数据的透传，可向通话方传输二进制数据。

推流管理

SDK 可以根据观看需求来动态管理对端推流状态。

视频延时和链路状态报告

SDK 可以监测视频延时和视频状态，并通过回调接口报告对应的状态数据。

基本概念

设备 ID

分为现场设备 ID 和远端设备 ID，用于标识设备身份，设备 ID 具有唯一性。如果网络中有相同设备 ID 的设备连网，会出现 MQTT 信令反复被踢下线又重连的情况，影响正常使用。

视频流 ID

用于标识现场设备的视频流编号，编号从0开始递增，与 json 配置中的 stream_config 数组元素编号一致。远端设备拉取视频流时，会通过现场设备 ID+ 视频流 ID 来指定要拉取的视频流。

接收句柄（conn_fd）

用于标识远端设备接收视频流的视频流接收器，编号从0开始递增；视频流接收器，用于桥接现场设备视频流和本地视频渲染窗口，拉取视频流时，需要指定对应的视频流接收器来收流；如下是收取视频流的过程。
现场设备视频流 >> 远端设备视频流接收器 >> 视频渲染窗口

视频窗口句柄

用于标识视频窗口，属于指向视频窗口的指针，例如 win32 的 HWND。通过视频窗口句柄，SDK 可以通过在指定视频窗口上进行画面渲染。

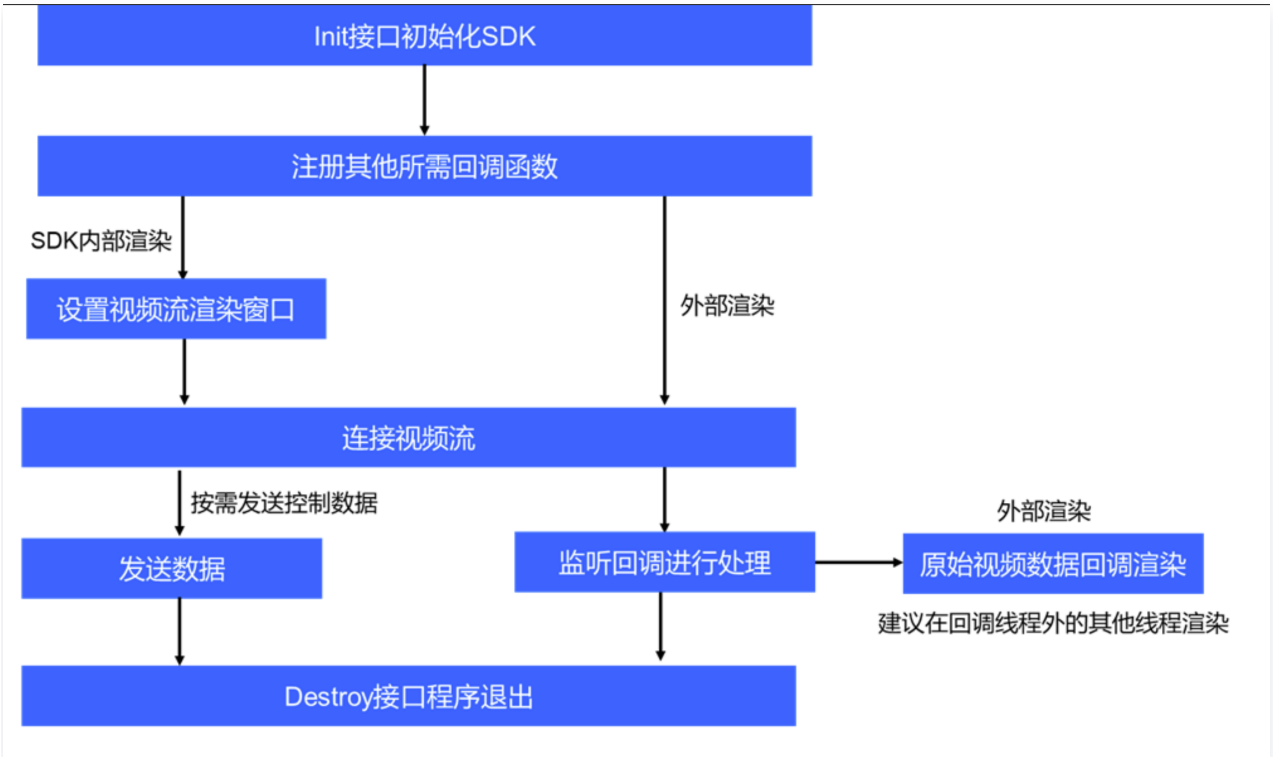
配置文件

config.json 文件，内含 SDK 的初始化配置。

SDK API 调用流程

最近更新时间：2024-03-29 17:10:21

SDK API 调用流程



调用步骤建议

> **注意：**
> signalStae 信令回调注册需要在初始化函数之前调用，其他回调函数可以在初始化之后注册。

步骤1：加载配置文件初始化

注册 signalState 信令回调接口，通过 Json 文件加载或输入 Json 字符串加载等方式，调用 Init 接口初始化 SDK。

步骤2：注册所需的回调函数

注册所需的回调函数。根据需求注册控制消息回调、延迟信息回调、日志回调、连接状态回调、远端设备视频链路信息回调、现场设备链路信息回调、原始视频数据回调接口等函数，处理对应的状态信息和事件。

步骤3：设置视频流渲染窗口

如果使用内置渲染，调用设置渲染窗口接口，绑定视频接收器 fd 和要渲染的视频窗口句柄。

步骤4：连接视频流

调用 connect 接口，连接指定现场设备 ID 的指定编号视频流 。当需要切换或断开视频时，可调用 disconnect 接口或 connect 接口。

步骤5：按需调用发送函数

当有外部二进制数据发送需求时，在所需发送时刻，调用对应的发送接口输入要发送的二进制控制数据。

步骤6：按需退出

当程序退出时，调用销毁接口销毁 SDK，C# 退出需要主动调用销毁接口。

Sample 代码示意

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "trro_remote.h"

//该代码sample未包含UI渲染部分，仅用于视频连接过程参考
int main() {
    //注册signal回调
    TRRO_registerSignalStateCallback(nullptr, [](void *context, SignalState state) {
        if(state == kTrroReady) {
            //服务器连接成功
            printf("init success \n");
        }
        if (state == kTrroAuthFailed) {
            //鉴权失败
            printf("device_id or password is incorrect\n");
        }
        if (state == kTrroKickout) {
            //被其他用户剔除
            printf("the device is kicked by server, may be there is another device using
the same device id\n");
        }
    });

    // 可按需替换为其他初始化函数，如使用配置字符串初始化
    int ret = TRRO_initJsonPath("./config.json");
    if(TRRO_SUCCED != ret) {
        if (ret == -TRRO_SIGNAL_CONNECT_OUTTIME) {
            printf("init process: wait for connecting\n");
        } else {
            printf("init fail ret %d\n", ret);
```

```
        }
    }

    //按需注册回调
    //注册视频连接状态回调
    TRRO_registerOnState(this, [](void* context, const char* gwid, int stream_id, int
conn_fd, int state) {
        printf("###### onstate gwid %s, streamid %d  conn_fd %d  state %d\n", gwid,
stream_id, conn_fd, state);
    });
    //注册媒体传输状态回调
    TRRO_registerOnMediaState(this, [](void* context, int windows_id, int fps, int bps,
int rtt, long long decd, int jitter,
        long long packets_lost, long long packets_received, int stun) {
            printf("onMediaState window id %d fps %d bps %d rtt %d decd %lld rb %d lost
%lld rec %lld stun %d\n",windows_id, fps, bps, rtt, decd, jitter, packets_lost,
packets_received, stun);
    });
    //注册延迟信息回调
    TRRO_registerAllLatencyCallback(this, [](void* context, const char* gwid, int
windows_id, long long latency1, long long latency2, long long gateway_time, int vcct)
{
        printf("allLatency %s, %d, latency1 %lld latency2 %lld video time %lld, vcct
%d\n", gwid, windows_id, latency1, latency2, gateway_time, vcct);
    });

    //注册原始视频数据回调，可用于外部渲染
    TRRO_registerRemoteFrameCallback(this, [](void * context, const char * gwid, int
stream_id, int conn_id, const char* data, int width, int height, long long videotime) {
        //可以在这里调用渲染函数播放视频数据，但不要阻塞处理，处理延迟大请异步调用其他线程
处理，否会导致视频卡顿
        //showMyVideo(gwid, conn_id, (unsigned char*)data, width, height);
    });

    //视频接收句柄数组
    int confds[8] = {0,1,2,3,4,5,6,7};
    //渲染窗口句柄数组
    WindowIdType window_hwnds[8] = { nullptr};
    //SDK内部渲染设置目标渲染窗口win32 HWND窗口句柄即可，外部渲染可设置空窗口句柄
(nullptr)，通过实现showMyVideo进行渲染
    TRRO_setWindows(confds, window_hwnds, 8);
    //连接视频流，假设每个现场设备需要连接4个视频流
    int stream_num = 4;
    int stream_indexs = [0,1,2,3];
    //连接现场设备1的4个视频流0-3，使用视频接收句柄 0-3
```

```
    TRRO_connect("field_deviceId1", "", stream_num , &stream_indexs[0], &confds[0]);
    //连接现场设备2的4个视频流0-3，使用视频接收句柄 4-7
    TRRO_connect("field_deviceId2", "", stream_num , &stream_indexs[0], &confds[4]);
    //等待退出
    while(true){
        sleep(3000);
        //断开视频流
        TRRO_disconnectAll();
        //退出SDK,释放资源
        TRRO_destroy();
        break;
    }
    return 0;
}
```

C/C++ 远端设备 SDK API

最近更新时间：2024-05-15 17:09:01

API 描述

适用于 Windows/Linux 64 位系统。头文件引用 trro_remote.h，API 可参考头文件中接口注释。下面列举常用接口描述。

API 概览

初始化接口

函数列表	描述
TRRO_initJsonPath	使用 json 配置文件路径初始化
TRRO_initJson	使用 json 字符串初始化

销毁接口

函数列表	描述
TRRO_destroy	销毁 SDK

状态获取类接口

函数列表	描述
TRRO_getGwList	获取可操控在线现场设备列表

视频操作接口

函数列表	描述
TRRO_setWindows	设置视频流渲染窗口
TRRO_connect	连接视频流
TRRO_disconnect	关闭指定视频连接
TRRO_disconnectAll	关闭所有视频连接
TRRO_SetRenderConfig	设置渲染配置

TRRO_fieldDeviceEncodeConfig	更新现场设备目标视频流编码参数

消息类接口

函数列表	描述
TRRO_sendControlData	向现场设备发送控制数据
TRRO_registerReportDataCallback	监听现场设备发送来的数据

注册类接口

函数列表	描述
TRRO_registerLatencyCallback	注册时延回调（废弃）
TRRO_registerAllLatencyCallback	注册时延回调（新）
TRRO_registerVideotimeCallback	注册视频帧采集时间戳回调（废弃）
TRRO_registerRemoteFrameCallback	注册原始视频图像帧回调
TRRO_registerRemoteEncodedFrameCallback	注册编码视频图像帧回调
TRRO_registerOnState	注册视频流链接状态回调
TRRO_registerOnMediaStat	注册视频流传输状态回调
TRRO_registerOnMediaStateInfo	注册视频流传输状态回调
TRRO_registerFieldSideNetworkState	注册域控设备网络状态检测回调
TRRO_registerSignalStateCallback	注册信令服务连接状态回调

获取软件信息

函数列表	描述
TRRO_getVersion	获取版本号

音频控制接口

函数列表	描述
TRRO_audioMute	静音现场设备

诊断类接口

函数列表	描述
TRRO_registerLogCallback	注册日志回调
TRRO_diagRequest	诊断视频流
TRRO_registerOnDiagReport	注册诊断信息回调接口

权限控制接口

函数列表	描述
TRRO_requestPermission	请求控制权限
TRRO_registerOnOperationPermissionState	注册现场设备权限状态

图像转换接口

函数列表	描述
TRRO_I4202ARGB	I420转 ARGB

初始化

使用说明：可根据配置输入类型（文件/字符串），选择对应初始化接口初始化，只需使用一个。

使用 json 配置文件路径初始化

```
/*
* @name : TRRO_initJsonPath
* @brief : SDK初始化，读取指定路径文件初始化， 接口为同步阻塞模式，等待信令连接成功后返回
* @input : jsonPath  json文件路径
* @return : 成功 1 失败 <= 0
*/
int TRRO_initJsonPath(const char* jsonPath);
```

参数	含义
jsonPath	json 配置文件路径
返回值	• 成功：1

	• 失败：<= 0

使用 json 字符串初始化

```
/*
* @name :  TRRO_initJson
* @brief : SDK初始化，读取输入JSON字符串初始化， 接口为同步阻塞模式，等待信令连接成功后返回
* @input : json   json配置字符串
* @return : 成功 1 失败 <= 0
*/
int TRRO_initJson(const char* json);
```

参数	含义
json	配置信息 json 字符串
返回值	• 成功：1 • 失败：<= 0

获取可操控在线现场设备列表

使用说明：可通过该接口获取到远端设备可以操控的在线现场设备列表，可根据实际需要使用。

```
/*
* @name :  TRRO_getGwList
* @brief : 获取在线现场设备设备列表，接口为同步阻塞模式
* @input : void
* @return : json字符串
*          {"ret":0,"msg":"suc","gateways":
[{"deviceID":"xx","name":"xx","type":"gateway","status":"connected","streams":4,"timestamp":1682231838673,"version":"xx"}],"count":1}
*/
const char* TRRO_getGwList();
```

参数	含义
返回值	现场设备列表 json 字符串

连接视频流

使用说明：可通过该接口对现场设备视频流进行连接。

```
/*
* @name : TRRO_connect
* @brief : 发起视频连接，可多次调用连接不同流，异步模式，根据 onState 状态回调确认视频连接成功
* @input : gwid 目标连接的现场设备 ID
*  record_config:
*  优先使用录制配置，当默认录制配置命名规则无法满足需求时使用，json 字符串，需要对每一路进行配置 eg:"{\"file_names\": [{\"file\":\"test\", \"duration\":15}, {\"file\":\"test01\", \"duration\":15}]}" ps:file 文件名 duration 分片时长单位秒
*  streams_num 要拉取现场设备视频流的个数， 与conn_fds和streams_num数组长度匹配， 值的范围 1 到 现场设备支持的device_streams数量
*  streams_id 现场设备视频流的 ID 数组， 现场设备视频流 ID 从0开始， 最大值为现场设备支持的 device_streams数量 -1
*  conn_fds conn_fd数组，conn_fd为接收视频流的句柄标识，自行编号，取值从0开始，最大值为 max_streams -1，max_streams在远端设备配置文件配置
* @return : 成功 1 失败 <= 0
*/
int TRRO_connect(const char* gwid, const char* record_config, int streams_num, int* streams_id, int* conn_fds);
```

参数	含义
gwid	现场设备 ID
record_config	视频录制配置，建议当默认录制配置无法满足需求时使用，json 字符串，需要对每一路进行配置，例如： `{ "file_names": [{ "file": "test_%D_%T", "duration": 15 }, { "file": "test01_%D_%T", "duration": 15 }] }` 参数： • file：文件名, 支持%D自动补充日期，%T自动补充时间 • duration：切片时长（单位：秒）

streams_num	要连接视频流的数量
streams_id	要连接的现场设备视频流 ID 数组
conn_fds	接收视频流使用的接收视频流句柄数组
返回值	• 成功：1 • 失败：<= 0

设置视频流渲染窗口

使用说明：可通过该接口设置渲染窗口，内部渲染时使用，外部渲染时设置空指针即可。

```
/*
* @name :  TRRO_setWindows
* @brief : 设置接收流句柄对应的显示窗口句柄
* @input : conn_fds 接收视频流句柄数组
*        windows  显示窗口句柄数组， 显示窗口句柄为显示窗体的句柄/指针，如 Windows 平台的 HWND;
*                 外部渲染时，数组中的显示窗口句柄配置为 nullptr
*        num      设置显示窗口数量，与 conn_fds 和 windows 数组长度匹配
* @return :void
*/
void TRRO_setWindows(int* conn_fds, WindowIdType * windows, int num);
```

参数	含义
conn_fds	接收视频流句柄数组
windows	显示窗口句柄数组，外部渲染时可设置为空指针数组
num	与 conn_fds 和 windows 数组长度一致

关闭视频连接

使用说明：可通过该接口断开视频连接。

关闭指定视频连接

```
/*
* @name :  TRRO_disconnect
* @brief : 关闭conn_fds 对应的视频连接
* @input : conn_fds 要关闭视频连接对应的视频接收句柄数组
```

```
*          fd_num conn_fd数组长度
* @return : 成功 1 失败 <= 0
*/
int TRRO_disconnect(int* conn_fds, int fd_num);
```

参数	含义
conn_fd	视频接收句柄数组
fd_num	数组长度
返回值	成功 1 失败 <= 0

关闭所有视频连接

```
/*
* @name :  TRRO_disconnectAll
* @brief : 关闭所有视频连接
* @input : void
* @return : 成功 1 失败 <= 0
*/
int TRRO_disconnectAll();
```

向现场设备发送控制数据

使用说明：可通过该接口给现场设备发送数据。目前限制单次发送数据大小700字节，频率限制100次/秒。

```
/*
* @name :  TRRO_sendControlData
* @brief : 向现场设备发送控制数据
* @input : gwid     目标现场设备 ID
*          msg      发送消息，二进制透传
*          len      消息长度
*          qos      消息传输qos  0:不可靠传输 1:可靠传输
* @return : 成功 1 失败 <= 0
*/
extern "C" TRRO_EXPORT int TRRO_sendControlData(const char* gwid, const char*
msg, int len, int qos = 0);
```

参数	含义

gwid	现场设备 ID
msg	发送二进制数据
len	消息长度
qos	发送 qos： • 0：不可靠传输 • 1：可靠传输
返回值	成功：1 失败：<= 0

注册回调函数

使用说明：可注册时延、现场设备数据、视频帧图像数据、视频连接状态和视频传输状态等回调，获取对应信息。

> ⚠ **注意：**
> 注册的回调函数，回调函数实现中请勿在回调线程长时间阻塞，若需要长时间处理建议转移到其他线程处理。

注册延时回调函数

```
extern "C" TRRO_EXPORT void TRRO_registerLatencyCallback(void* context,
TRRO_onLatencyReport * callback);
extern "C" TRRO_EXPORT void TRRO_registerAllLatencyCallback(void* context,
TRRO_onAllLatencyReport * callback);

/*
* @name :  TRRO_onLatencyReport (depreciated)
* @brief : 视频时延回调, 旧接口，建议使用onAllLatencyReport接口获取
* @input : context     回调上下文指针, 返回注册回调函数时传入的context
*          gwid        视频来源现场设备id
*          conn_fd     对应视频流接收句柄
*          latency     返回视频上行时延ms
* @return : void
*/
typedef void STD_CALL TRRO_onLatencyReport(void* context, const char* gwid, int
conn_fd, long long latency);


/*
* @name :  TRRO_onAllLatencyReport
* @brief : 时延回调
```

```
* @input : context    回调上下文指针, 返回注册回调函数时传入的 context
*         gwid       视频来源现场设备 ID
*         conn_fd    对应视频流接收句柄
*         latency1   返回均值估计视频时延ms
*         latency2   返回最大估计视频时延ms
*         videotime  当前视频帧的现场设备采集时间戳
*         rcct       控制信道往返时延ms, 等于控制上行延迟+控制下行延迟
* @return : void
*/
typedef void STD_CALL TRRO_onAllLatencyReport(void* context, const char* gwid, int
conn_fd, long long latency1, long long latency2, long long videotime, int rcct);
```

注册视频帧时间戳回调函数

```
extern "C" TRRO_EXPORT void TRRO_registerVideotimeCallback(void* context,
TRRO_onVideotimeReport * callback);
/*
* @name :  TRRO_onVideotimeReport (depreciated)
* @brief : 视频帧采集时间戳回调, 旧接口，建议使用 onAllLatencyReport 接口获取
* @input : context    回调上下文指针, 返回注册回调函数时传入的 context
*         gwid       视频来源现场设备 ID
*         conn_fd    对应视频流接收句柄
*         videotime  当前视频帧的现场设备采集时间戳
* @return : void
*/
typedef void STD_CALL TRRO_onVideotimeReport(void* context, const char* gwid, int
conn_fd, long long videotime);
```

注册现场设备消息回调函数

```
extern "C" TRRO_EXPORT void TRRO_registerReportDataCallback(void* context,
TRRO_onReportData * callback);

/*
* @name :  TRRO_onReportData
* @brief : 接收来自现场设备信息回调
* @input : context    回调上下文指针, 返回注册回调函数时传入的 context
*         gwid       消息来源现场设备 ID
*         msg        消息, 二进制透传
```

```
*        len        消息长度
*        qos        消息来源传输 qos， 0:不可靠传输, 1:可靠传输
* @return : void
*/
typedef void STD_CALL TRRO_onReportData(void* context, const char* gwid, const
char* msg, int len, int qos);
```

注册视频帧回调函数

```
extern "C" TRRO_EXPORT void TRRO_registerRemoteFrameCallback(void* context,
TRRO_onRemoteFrameData * callback, int frame_type = 0);
extern "C" TRRO_EXPORT void TRRO_registerRemoteEncodedFrameCallback(void*
context, TRRO_onRemoteEncodedFrameData* callback);

/*
* @name :  TRRO_onRemoteFrameData
* @brief : 原始视频图像帧回调
* @input : context     回调上下文指针, 返回注册回调函数时传入的 context
*        gwid        视频帧的来源现场设备 ID
*        stream_id   视频帧的来源现场设备流 ID
*        conn_fd     接收视频流的句柄标识
*        data        视频图像帧数据，数据格式为YUVI420，可通过I4202ARGB接口转 ARGB
*        width       视频帧宽
*        height      视频帧高
*        videotime   当前视频帧的现场设备采集时间戳
* @return : void
*/
typedef void STD_CALL TRRO_onRemoteFrameData(void* context, const char* gwid,
int stream_id, int conn_fd, const char* data, int width, int height, long long videotime);


/*
 * @name :  TRRO_onRemoteEncodedFrameData
 * @brief : 编码视频图像帧回调
 * @input : context      回调上下文指针, 返回注册回调函数时传入的 context
 *        gwid         视频帧的来源现场设备 ID
 *        stream_id    视频帧的来源现场设备流 ID
 *        conn_fd      接收视频流的句柄标识
 *        data         编码的视频图像帧数据
 *        len          编码的视频图像帧的长度
 *        trro_codec   编码的 codec 类型
 *        is_key_frame 是否是关键帧
 *        videotime    当前视频帧的现场设备采集时间戳
 * @return : void
```

```
 */
typedef void STD_CALL TRRO_onRemoteEncodedFrameData(void* context, const
char* gwid, int stream_id, int conn_fd,
                                        const char* data, int len, TrroCodec trro_codec, bool
is_key_frame,
                                        long long videotime);
```

注册状态回调函数

```
extern "C" TRRO_EXPORT void TRRO_registerOnState(void* context, TRRO_OnState *
callback);
extern "C" TRRO_EXPORT void TRRO_registerOnMediaState(void* context,
TRRO_OnMediaState * callback);
extern "C" TRRO_EXPORT void TRRO_registerOnMediaStateInfo(void* context,
TRRO_OnMediaStateInfo * callback);
extern "C" TRRO_EXPORT void TRRO_registerFieldSideNetworkState(void* context,
TRRO_onFieldSideMediaState * callback);
extern "C" TRRO_EXPORT int TRRO_registerSignalStateCallback(void *context,
TRRO_onSignalState *callback);

//视频连接状态
enum TrroState {
    kDisconnect = 0,
    kConnecting = 1,
    kConnected = 2,
    kDisconnecting = 3,
};

/**
 * SignalState 信令连接状态枚举
 */
enum SignalState {
    kTrroReady = 0, /**< 连接建立成功 */
    kTrroLost = 1,  /**< 连接断开，内部会进行自动重连 */
    kTrroReup = 2,  /**< 自动重连成功 */
    kTrroKickout = 3,
    kTrroAuthFailed = 4,  /**< 用户名或者密码错误 */
};


/* TRRO_OnMediaStateInfo 使用该结构体 包含媒体传输的详细信息
* fps         每秒帧数目
```

```
* bps        每秒数据量
* rtt        UDP ping 网络往返时间
* decd       解码耗时ms
* jitter     接收抖动值
* packets_lost 丢包率（0-255），丢包百分比 = packets_lost / 255 * 100%
* packets_received 总接收数目
* stun       穿网模式 0：host, 1：srflx, 2：prflx, 3：relay
* 单位(%) 视频播放卡顿率 = 视频播放的累计卡顿时长 / 视频播放的总时长。
* lag_k100    视频帧间隔超过100ms视为卡顿，并将其计入视频播放的累计卡顿时长，最终计算
获得的卡顿率。
* lag_k150    视频帧间隔超过150ms视为卡顿，并将其计入视频播放的累计卡顿时长，最终计算
获得的卡顿率。
* duration    视频播放的总时长
*/
struct TrroMediaState {
    int fps;
    int bps;
    int rtt;
    long long decd;
    int jitter;
    long long packets_lost;
    long long packets_received;
    int stun;
    double lag_k100;
    double lag_k150;
    long long duration;
};



/*
* @name :  TRRO_OnState
* @brief : 视频流连接状态回调
* @input : context    回调上下文指针, 返回注册回调函数时传入的 context
*          gwid       视频帧的来源现场设备 ID
*          stream_id  视频帧的来源现场设备流 ID
*          conn_fd    接收视频流的句柄标识
*          state      视频流连接状态，对应 TrroState 状态
* @return : void
*/
typedef void STD_CALL TRRO_OnState(void* context, const char* gwid, int stream_id,
int conn_fd, int state);

/*
* @name :  TRRO_OnMediaState
```

```
* @brief : 视频流传输状态回调
* @input : context     回调上下文指针, 返回注册回调函数时传入的 context
*          conn_fd     接收视频流的句柄标识
*          fps         每秒帧数目
*          bps         每秒数据量
*          rtt         UDP ping 网络往返时间
*          decd        解帧耗时ms
*          jitter      接收抖动值
*          packets_lost 丢包率（0-255），丢包百分比 = packets_lost / 255 * 100%
*          packets_received 总接收数目
*          stun        穿网模式 0：host, 1：srflx, 2：prflx, 3：relay
* @return : void
*/
typedef void STD_CALL TRRO_OnMediaState(void* context, int conn_fd, int fps, int
bps, int rtt, long long decd, int jitter,
    long long packets_lost, long long packets_received, int stun);


/*
* @name :  TRRO_OnMediaStateInfo
* @brief : 视频流传输状态回调
* @input : context     回调上下文指针, 返回注册回调函数时传入的 context
*          conn_fd     接收视频流的句柄标识
*          mediastate  媒体相关的详细信息
* @return : void
*/
typedef void STD_CALL TRRO_OnMediaStateInfo(void* context, int conn_fd,
TrroMediaState mediastate);



/**
* @name : TRRO_onFieldSideMediaState
* @brief : field端网络状态信息，server模式时有意义
* @input :      context          上下文
*             gwid             field端ID
*             stream_id        field端流ID
*             fps              field端推送帧率
*             bps              field端推送码率
*             rtt              field端封包来回时间
*             jitter           field端抖动
*             lost             field端丢包率
* @return : void
*/
```

```
typedef void TRRO_onFieldSideMediaState(void* context, const char* gwid, int
stream_id, int fps, int bps, int rtt, int jitter, int lost);

/**
* @name TRRO_onSignalState
* @brief 信令连接状态回调
* @param[in]    context      上下文
* @param[in]    state        信令连接状态，参考枚举SignalState
* @return void
*/
typedef void TRRO_onSignalState(void *context, SignalState state);
```

注册权限控制回调

```
extern "C" TRRO_EXPORT void TRRO_registerOnOperationPermissionState(void*
context, TRRO_OnOperationPermissionState* callback);


//操控权限
enum TrroPermission {
    kPermissionGuest = 0,
    kPermissionMaster = 1,
};

/*
* #name :   TRRO_OnOperationPermissionState
* @brief :  回调现场设备操控权限状态通知
* @input :  context           回调上下文指针, 返回注册回调函数时传入的 context
*           field_devid       来源现场设备 ID
*           self_permission   本设备当前的操控权限，参考 TrroPermission， 0 是 guest，只有
观看权限， 1是 master，拥有完全控制权限
*           master_devid      拥有 master 权限的远端设备 ID
* @return : void
*/
typedef void STD_CALL TRRO_OnOperationPermissionState(void* context, const char*
field_devid, int self_permission, const char* master_devid);
```

注册诊断类接口

```
extern "C" TRRO_EXPORT int TRRO_registerLogCallback(void* context,
TRRO_OnLogData * callback);
```

```
extern "C" TRRO_EXPORT void TRRO_registerOnDiagReport(void* context,
TRRO_OnDiagReport * callback);

//日志等级
enum TrroLogLevel {
    LOG_INFO = 1,
    LOG_WARNING = 2,
    LOG_ERROR = 3,
};


/*
* @name :   TRRO_OnLogData
* @brief :  日志回调
* @input :  context       回调上下文指针, 返回注册回调函数时传入的 context
*           msg           日志内容
*           level         日志级别,参考枚举 TrroLogLevel
* @return : void
*/
typedef void STD_CALL TRRO_OnLogData(void* context, const char* msg, int level);


/*
* #name :   TRRO_OnDiagReport
* @brief :  诊断信息回调
* @input :  context     回调上下文指针, 返回注册回调函数时传入的 context
*           gwid        发起诊断的目标现场设备 ID
*           type        1 成功 , < 0 相关错误信息
*           json        回调诊断详细信息 json 格式
* @return : void
*/
typedef void STD_CALL TRRO_OnDiagReport(void* context, const char* gwid, int type,
const char* json);
```

获取版本号

```
//获取SDK版本号
const char * TRRO_getVersion();
```

销毁 SDK

使用说明：可通过该接口销毁 SDK，释放资源，可用于程序退出时需要主动释放底层资源场景（例如 C# 程序）。

```
//销毁SDK
extern "C" TRRO_EXPORT void TRRO_destroy();
```

静音现场设备

使用说明：将目标现场设备静音，仅支持 server 模式。

```
/*
* @name :  TRRO_audioMute
* @brief : Mute/unMute 目标现场设备的音频 仅 Server 模式
* @input : gwid     目标现场设备 ID
*          mute     true 静音， false 取消静音
* @return : 成功 1 失败 <= 0
*/
extern "C" TRRO_EXPORT int TRRO_audioMute(const char* gwid, bool mute);
```

参数	含义
gwid	目标现场设备 ID
mute	• true：静音 • false：取消静音

诊断视频流

使用说明：用于诊断视频状态，调用TRRO_connect接口后，如获取不到视频流，可通过调用诊断接口获取诊断报告。

```
/*
* @name : TRRO_DiagRequest
* @brief : 需要主动出发，诊断网关、控制端当前状态输出报告
* @input : gwid     目标设备 ID
* @return : 成功1 失败 <= 0
*/
extern "C" TRRO_EXPORT int TRRO_diagRequest(const char* gwid);
```

参数	含义
gwid	目标现场设备 ID

请求控制权限

使用说明：向现场设备请求控制权限，请求成功后才可向对应现场设备发送及接收控制数据。

```
//操控权限
enum TrroPermission {
    kPermissionGuest = 0,
    kPermissionMaster = 1,
};

/*
* @name : TRRO_requestPermission
* @brief : 向网关发出权限请求，网关会反馈TRRO_OnOperationPermissionState 更新权限
* @input : gwid         目标设备 ID
*          permisson    参考结构体 TrroPermission
* @return : 成功1 失败 <= 0
*/
extern "C" TRRO_EXPORT int TRRO_requestPermission(const char* gwid, int
permisson);
```

参数	含义
gwid	目标现场设备 ID
permisson	控制权限

I420转ARGB

使用说明：用于图像数据的格式转换，将 yuvi420格式转为 ARGB，支持分辨率 resize。

```
/*
* @name :  TRRO_I4202ARGB
* @brief : YUV I420 图像格式转 ARGB 图像格式，转换时带 resize 功能
* @input : argb     存放输出A RGB 图像数据的指针，ARGB 的 int 表示： 0XFF(RR)(GG)(BB)
 ，  RR GG BB 为对应的 rgb 的值
*          yuvI420  输入 yuvI420 图像数据指针
*          src_width  输入图像宽度
*          src_height  输入图像高度
*          dst_width   输出图像宽度
*          dst_height  输出图像高度
* @return : 成功 1 失败 <= 0
```

```
*/
extern "C" TRRO_EXPORT int TRRO_I4202ARGB(int* argb, char* yuvI420, int src_width,
int src_height, int dst_width, int dst_height);
```

参数	含义
argb	输出的 argb 数据
yuvI420	输入的 yuvi420格式数据
src_width	输入图像宽度
src_height	输入图像高度
dst_width	输出图像宽度
dst_height	输出图像高度

设置渲染配置

使用说明：设置接收流句柄对应的内部渲染模式, 仅内部渲染时使用。

```
/*
* @name :  TRRO_SetRenderConfig
* @brief : 设置接收流句柄对应的内部渲染模式, 仅内部渲染时使用
* @input : conn_fds 要设置渲染模式的接收视频流句柄数组
*          rotation 旋转角度 数组 rotation 为正顺时针旋转，rotation为负 逆时针旋转
*          scale 分辨率匹配模式数组 0 窗口拉伸， 1 保持原比例
*          config 预留渲染配置数组 0xFFFFFFF1 镜像渲染 0xFFFFFFF0 关闭镜像渲染
*          num  配置的内部渲染流数量，与输入数组长度匹配
* @return : 成功 1 失败 <= 0
*/
extern "C" TRRO_EXPORT int TRRO_SetRenderConfig(int *conn_fds, int *rotation, int
*scale, int *config, int num);
```

参数	含义
conn_fds	要设置渲染模式的接收视频流句柄数组
rotation	旋转角度，数组 rotation 为正顺时针旋转，rotation为负逆时针旋转
scale	分辨率匹配模式数组 0 窗口拉伸， 1 保持原比例

config	预留渲染配置数组 0xFFFFFFF1 镜像渲染 0xFFFFFFF0 关闭镜像渲染
num	配置的内部渲染流数量，与输入数组长度匹配

更新现场设备目标视频流编码参数

使用说明：设置接收流句柄对应的内部渲染模式, 仅内部渲染时使用。

```
/*
* @name :  TRRO_fieldDeviceEncodeConfig
* @brief : 更新现场设备目标视频流编码参数
* @input : gwid 现场设备id
*          streams_id 目标视频流编号
*          encode_config 待更新的编码参数，json格式字符串，缺省字段将保持当前值不进行更新。 下面是更新支持的编码参数示意
*  {
*      "fps": 30,
*      "encode_width": 1920,
*      "encode_height": 1080,
*      "bps": 3000,
*      "min_fps": 30,
*      "min_bps": 1800,
*      "force_min": 0,
*      "min_width": 1920
*  }
*/
extern "C" TRRO_EXPORT int TRRO_fieldDeviceEncodeConfig(const char* gwid, int streams_id, const char* encode_config);
```

参数	含义
gwid	现场设备 ID
streams_id	目标视频流编号
encode_config	待更新的编码参数，json格式字符串，缺省字段将保持当前值不进行更新。 下面是更新支持的编码参数示意

远端设备配置说明

最近更新时间：2024-02-20 10:59:32

JSON 配置文件样例

配置文件简述
文件名：config.json
文件位置：${workspace}/config.json
文件类型：json
注意："//"后注释在使用时要删除。最小节点如下

```
{
   "device_id":"dev1",  //修改为控制台中创建的远端设备ID
   "device_name":"vin99",
   "max_streams":8,
   "cloud_mode":"public",
   "certificate":"./device.pem",
   "projectid" : "xxxxx", //修改为控制台中创建的项目ID
   "password": "xxxxx" //修改为控制台中创建的密码
}
```

JSON 配置节点含义

节点名	类型	含义	默认值	备注	参数范围
device_id	字符串型（String）	设备 ID，用于注册到服务器的唯一编码	-	必选	-
device_name	字符串型（String）	设备名称，用于辨识设备信息	-	必选	-
max_streams	数字型（Number）	远端设备连接的最大流数目	-	必选	建议填8
projectid	字符串型（String）	项目 ID	-	公有云必填	-
cloud_mode	字符串型（String）	云模式	-	必填	• public：公有云 • private：私有云

password	**字符串型（String）**	设备密钥	-	公有云必填	-
certificate	**字符串型（String）**	服务器公钥证书	-	公有云必填，device.pem 文件路径	-
force_login	数字型（Number）	是否强制登录	0	同名 ID 登录是否剔除之前的设备登录	• 0：已有相同 ID 登录时，当前设备登录被剔除 • 1： 已有相同 ID 登录时，当前设备会强制登录，将之前设备剔除
server_ip	**字符串型（String）**	TRRO 服务端 IP	-	公有云账号无需指定，私有化必填	-
server_port	**数字型（Number）**	TRRO 服务端端口号	-	公有云账号无需指定，私有化必填	-
log_enable	**布尔型（Boolean）**	日志开关	1	可选	• 0：不使能 • 1：info日志 • 2：info+debug日志
sdk_mode	**字符串型（String）**	流传输模式	"server"	可选	-
rtc_server_ip	**字符串型（String）**	媒体服务端 IP	-	公有云无需指定，私有化必填	-
rtc_server_port	**数字型（Number）**	媒体服务端端口号	-	公有云无需指定，私有化必填	-
data_transport	数字型（Number）	控制消息传输模式	0	可选	• 0：udp通道传输 • 2：tcp通道传输

audio_enable	数字型（Number）	音频采集使能	0	可选	• 0：不使能 • 1：使能
audio_receive	数字型（Number）	音频接收使能	1	可选	• 0：不使能 • 1：使能
auto_permission	数字型（Number）	是否自动请求权限	1	可选	• 0：应用请求权限 • 1：自动请求权限
play_mode	数字型（Number）	观看模式	0	可选	• 0：超低时延观看 • 1：流畅观看
decode_hw_acc	数字型（Number）	是否硬件解码优先	0	可选	• 0：软解 • 1：硬解优先
file_path	字符型（String）	保存文件路径	–	可选，配置时开启录制，需保证配置路径已存在	–
push_path	字符型（String）	转推视频流路径	–	可选，用于向指定地址转推 RTMP 视频流	当指定地址配置为 `rtmp://xxxxx/` 视频实际转推地址 `rtmp://xxxxx/{现场设备ID}/video_{流ID}`
min_port	数字型（Number）	视频传输本地端口段最小值	50000	可选	–
max_port	数字型（Number）	视频传输本地端口段最大值	50100	可选	–

错误码及排查

最近更新时间：2024-02-07 10:08:22

错误码的定义位于 trro_remote.h 头文件中。函数返回值为错误码负值，如错误码为16777215，返回值为-16777215。错误码具体含义及排查方法如下表所示：

模块	错误码定义	错误码16进制	错误码10进制	排查方向
通用错误码	TRRO_SUCCED	0x00000001	1	成功返回值
	TRRO_COMMON_ERROR	0x00FFFFFF	16777215	参数输入或初始化状态异常
	TRRO_GWID_NOT_FOUND	0x00000010	15728640	不存在的网关 ID
	TRRO_PARAM_INVALID	0x00F00000	16777215	参数非法
配置文件加载模块（检查配置文件）	TRRO_CONFIG_ERROR	0x01FFFFFF	33554431	通用配置类错误
	TRRO_CONFIG_PARSE_FAILED	0x01000002	16777218	配置文件的格式解析异常，建议使用在线 json 校验工具进行检查。
	TRRO_CONFIG_ILLEGAL	0x01000003	16777219	配置文件的参数非法，建议检查最小节点是否覆盖
	TRRO_CONFIG_UNEXIST	0x01000004	16777220	配置文件不存在
	TRRO_CONFIG_CER_FAILED	0x01000005	16777221	公有云模式下证书错误
初始化模块（检查配置文件和接口入参是否正常）	TRRO_INIT_ERROR	0x02FFFFFF	50331647	通用初始化失败，未知原因
	TRRO_INIT_INPUT_ILLEGAL	0x02000002	33554434	输入参数非法，需检查入参是否正常
	TRRO_INIT_PARSE_fAILED	0x02000003	33554435	初始化参数解析失败，检查配置文件最小节点是否正常

	TRRO_INIT_CREAT_MEDIAMODE_FAILED	0x02000004	33554436	媒体模块创建失败
	TRRO_INIT_LICENSE_CHECK_FAILED	0x02000005	33554437	LICENSE 校验失败
	TRRO_INIT_INVALUDE_INPUT	0x02000006	33554438	接口调用入参非法
	TRRO_INIT_MQTT_ABNORMAL	0x02000007	33554439	始化 mqtt 异常
	TRRO_INIT_REPEAT	0x02000008	33554440	重复初始化
信令服务器模块（检查服务器连接是否正常）	TRRO_SIGNAL_ERROR	0x03FFFFFF	67108863	信令模块异常
	TRRO_SIGNAL_REGIST_FAILED	0x03000002	50331650	信令注册失败
	TRRO_SIGNAL_STATUS_ABNORMAL	0x03000003	50331651	连接 MQTT 信令服务异常
	TRRO_SIGNAL_MESSAGE_FAILED	0x03000004	50331652	消息发送失败
	TRRO_SIGNAL_CONNECT_OUTTIME	0x03000005	50331653	连接 MQTT 服务超时
流传输模块（检查网络和防火墙）	TRRO_CONNECT_ERROR	0x05FFFFFF	100663295	连接异常
	TRRO_CONNECT_OUTTIME	0x05000002	83886082	超时断连
	TRRO_CONNECT_RESET	0x05000003	83886083	connect 的时候如果之前有未断开的连接需要重置
诊断工具模块	TRRO_DIAG_STARTED	0x06000000	100663296	诊断工具正在启动中
	TRRO_DIAG_TIMEOUT	0x06000001	100663297	诊断工具获取推流信息超时
	TRRO_DIAG_FAILED	0x06000002	100663298	诊断工具处理失败

权限配置模块	TRRO_UNAUTHORIZED	0x07000000	117440512	网关未授权无法发送消息
消息通道	TRRO_MESSAGE_BYTE_EXCEED	0x08000002	134217730	发送字节数超出限制