实时互动-工业能源版

开发指南

腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【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

开发指南
Demo 体验流程

最近更新时间：2024-02-07 10:08:21

申请测试

申请体验资格，单击实时互动-工业能源版 内测申请 。

创建项目信息

参见 控制台指南 相关文档，在控制台创建项目、设备 ID、设备密码。用于推流端和拉流端的 config.json 中账户信息配置。

申请试用授权

在 控制台 > 视频授权 中，单击申请试用，填写问卷后获取下发的试用授权。在控制台 > 设备管理中，选中创建的测试现场设备，单击管理操作，绑定试用授权。

Demo 下载地址

- 现场设备端
- 远端设备端

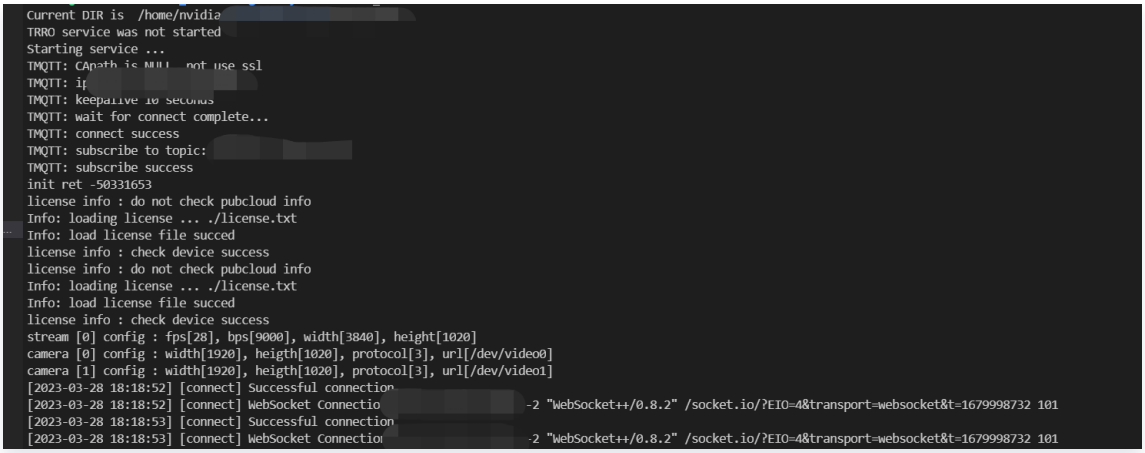
推流端 Demo 启动

1. 修改推流端 Demo 目录下， config.json 配置文件。根据 现场设备配置说明 ，修改 config.json 配置文件，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现场设备ID
 "device_name":"vin234",
 "device_streams":1, //如果是多路输入,修改这里的流数目，并增加streams_config数组中
的元素个数
 "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
 "streams_config": [
 {
    "fps":30,
    "bps":2000,
    "width":1920,
```

```
    "height":1080,
    "camera":0, //修改为实际接入的相机，对应/dev/videox
    "protocol":"v4l2" //如果为外部输入yuv数据，修改为outside。 如果为外部输入编码流数
据，修改为outenc
  }
 ]
}
```

2. 启动运行脚本 run_loader.sh，等待远端设备拉流，成功启动后，提示连接服务器成功。



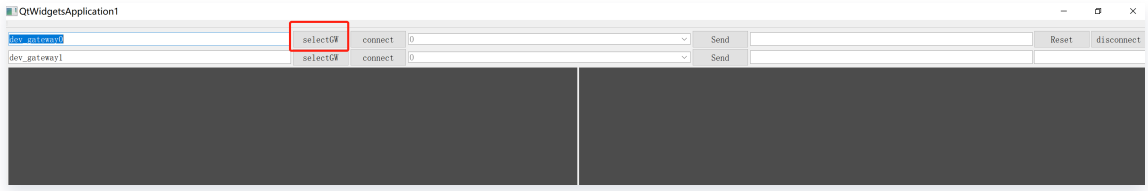
拉流端 Demo 启动

1. 修改拉流端 Demo 目录下，`config.json` 配置文件。根据 **远端设备配置说明**，修改 `config.json` 配置文件，注意："//"后注释在使用时要删除。

```
{
   "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
}
```

2. 启动拉流端程序：
   - Windows 端双击运行 `QtApplicationWidget1.exe`
   - Linux 端运行 `run.sh`：`./run.sh`

   单击 **select**，选择拉流的网关设备，并单击 **confirm**。



单击 connect 进行拉流。

视频观看与切流

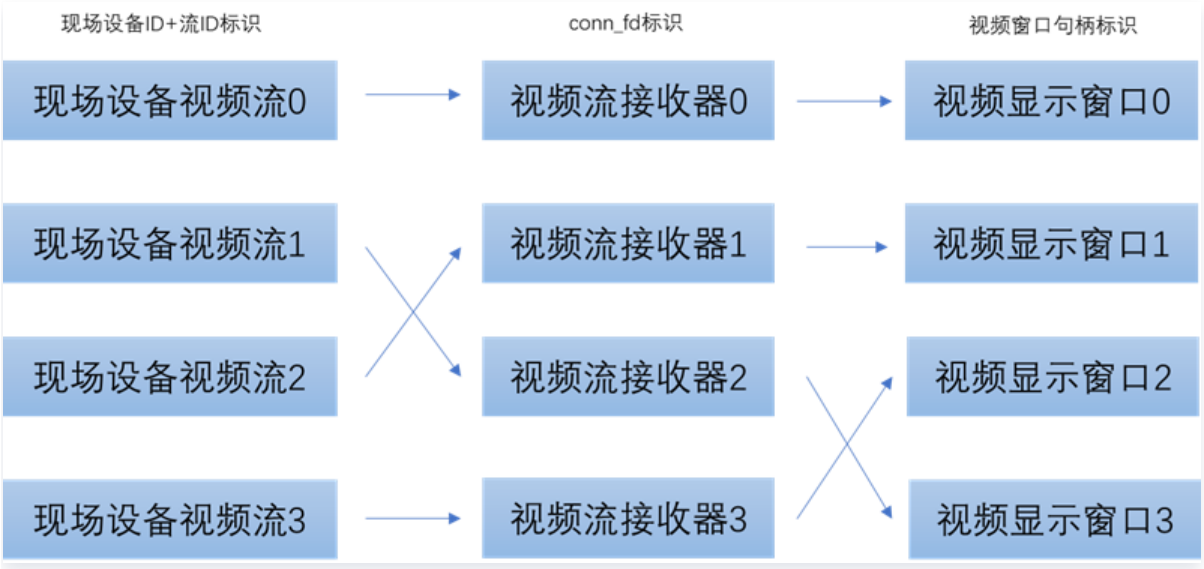
最近更新时间：2024-02-19 17:19:11

> **说明：**
> 视频观看与切流，用于控制远端设备上现场设备视频流的显示。目前支持对多个现场设备的多个视频流进行观看和切换。由远端设备 SDK 发起，无需对现场设备 SDK 进行操作。

用法介绍

视频流的接收

远端设备接收视频流时，调用 TRRO_connect 接口会通过视频流接收器来接收视频流，并关联到通过 TRRO_setWindows接口设置的视频显示窗口。下图是一组视频流和显示窗口关联关系示意：



- **现场设备视频流：**由现场设备 ID + 流 ID 唯一标识。流 ID 编号从0到 N，对应现场设备 streams_config 配置中视频流数组的元素编号。
- **视频显示窗口：**由视频窗口句柄唯一标识，例如 win32 的 HWND。内部渲染模式下，SDK 可根据视频窗口句柄在指定窗口上进行视频渲染显示。外部渲染模式下，由开发者自己维护视频显示窗口和视频的渲染显示。
- **远端设备视频流接收器：**由接收句柄（conn_fd）唯一标识。接收句柄编号从0开始递增，可由开发者自定义，一般可设置为窗口编号。

> **注意：**
> 同一时间，一个视频流接收器只能接收一个现场设备视频流。当视频流接收器连接新的视频流时，之前连接的视频流会自动断开。

可以看出，对于内部渲染，远端设备视频流接收器桥接了现场设备视频流和视频窗口，可让 SDK 自动完成视频流显示。对于外部渲染，远端设备视频流接收句柄可协助开发者来标识 SDK 回调的视频流数据应该在哪个窗口上进行渲

染。

视频流的切换

远端设备进行视频流切换时，可通过 TRRO_connect 接口进行。根据新切换的视频流是否要在新的窗口显示，可分两类场景：

1. 切换后视频流在已有窗口显示：

使用已有窗口对应的接收句柄，通过 TRRO_connect 接口接收该窗口要切换的视频流。这时新视频流会切换到对应窗口显示，而窗口上原有视频流会被断开。

2. 切换后视频流在新建窗口显示：

使用新窗口对应的接收句柄，通过 TRRO_connect 接口接收切换后的视频流，并通过 TRRO_disconnect 接口断开切换前的视频流。

视频窗口的切换

内部渲染模式下，远端设备进行视频窗口切换时，可通过 TRRO_setWindows 接口进行。一般在 SDK 启动阶段，会将视频流接收器的接收句柄和视频窗口进行关联。在窗口布局需要调整时（例如交换两个视频流的所在窗口，而不涉及视频流本身的切换），可通过 TRRO_setWindows 接口更新视频流接收器的接收句柄和视频窗口的对应关系，从而在不断连视频流的情况下，完成视频流窗口的切换。

相关接口

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和切换。

```
/*
* @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:
*                缺省时，优先使用全局录制配置，当默认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* s
treams_id, int* conn_fds);
```

参数	含义
gwid	现场设备 ID
record_config	视频录制配置。json 字符串，需要对连接的每一路视频流进行配置。建议缺省优先使用全局录制配置，无法满足时再使用该方式。 `{ "file_names": [{ "file": "test", "duration": 15 }, { "file": "test01", "duration": 15 }] }` 参数： • file：文件名 • 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();
```

语音对讲

最近更新时间：2024-02-07 10:08:21

> **说明：**
> 语音对讲，用于远端设备和现场设备进行语音交互。 语音传输会随操控会话建立自动建议。SDK 默认只打开了上行音频。如需使用双向音频，请准备相关音频硬件。增加相关配置并重启 SDK。

硬件设备准备

设备端	上行音频： 现场设备->远端设备	下行音频： 远端设备->现场设备
现场设备	麦克风	扬声器
远端设备	扬声器	麦克风

软件配置修改

对 Linux 现场设备 SDK 及 Windows 远端设备 SDK 配置节点 config.json 文件修改，开启音频开关并设置音频设备。

开启音频开关

开启音频接收，将audio_receive配置为1； 开启音频采集，将audio_enable配置为1.

设备端	上行音频： 现场设备->远端设备	下行音频： 远端设备->现场设备
现场设备	"audio_enable":1	"audio_receive":1
远端设备	"audio_receive":1	"audio_enable":1

设置音频设备

SDK默认使用系统默认的音频播放和采集设备，Linux平台下如果目标音频设备不是系统默认设备，可通过配置指定使用对应的音频设备。

配置项	配置说明	配置举例
音频播放设备	• `audio_play":"hw:#Card,#Device,#SubDevice` • 其中 #Card , #Device , #SubDevice 为声卡设备编号，Linux	例如，目标设备是Card 0， Device 0，SubDevice #0：`audio_play":"hw:0,0,0`

	平台可通过 `aplay -l` 查询目标设备的card、device、subdevice 的编号。	
音频采集设备	• `audio_record":"hw:#Card,#Device,#SubDevice` • 其中 #Card , #Device , #SubDevice 为声卡设备编号，Linux平台可通过 `arecord -l` 查询目标设备的card、device、subdevice 的编号	例如，目标设备是Card 0， Device 0，SubDevice #0：`audio_record":"hw:0,0,0`

示例：执行命令 "aplay -l ", 可以查看音频播放设备。

```
nvidia@nvidia-desktop:~$ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: tegrahdagalent1 [tegra-hda-galen-t194], device 3: HDMI 0 [HDMI 0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

如图使用的播放设备为"card 0, device 3, subdevice 0"， 则在配置文件 config.json 中增加配置节点"audio_play":"hw:0,3,0"。

重新启动软件

完成配置文件的修改后，重新启动推流和拉流端。

控制数据传输

最近更新时间：2024-02-07 10:08:21

说明：
控制数据传输，用于现场设备和远端设备之间的二进制数据通信。两端SDK均提供一个数据发送接口和一个数据接收回调接口。

注意：
- 现场设备 SDK，数据发送接口会向与该现场设备处于会话连接中的所有远端设备发送数据消息。
- 远端设备 SDK，数据发送接口可以向会话连接中的指定现场设备发送数据消息，仅当该远端设备拥有现场设备控制权时，才可发送成功。

现场设备：

向远端设备发送数据

说明：
该接口会向与该现场设备处于会话连接中的所有远端设备发送数据消息。 发送消息单条长度限制为700字节，发送频率限制为100Hz。

```
/**
* @name   TRRO_sendControlData
* @brief  向远端设备发送数据
* @param[in] msg          消息体
* @param[in] len          消息体长度
* @param[in] qos          0:unreliable 1:reliable
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_sendControlData(const char* msg, int len, int qos
= 0);
```

参数	含义
msg	消息内容
len	消息长度
qos	发送 qos 类型。

	• 0：非可靠传输，不会重传 • 1：可靠传输
返回值	• 成功：1 • 失败：<= 0

注册控制消息回调

```
/**
* @name TRRO_registerControlDataCallback
* @brief 注册远端设备消息回调函数
* @param[in]   context          上下文指针，回调时会返回该上下文指针
* @param[in]   callback         回调函数
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerControlDataCallback(void* context,
TRRO_onControlData * callback);
```

参数	含义
context	回调上下文
TRRO_onControlData	回调函数

回调函数定义

```
/**
* @name TRRO_onControlData
* @brief 接收远端设备消息回调
* @param[in]    context       上下文指针，返回注册时传入的context
* @param[in]    controller_id  远端设备ID
* @param[in]    msg            消息体内容
* @param[in]    len            消息体长度
* @param[in]    qos            消息qos类型 0:unreliable, 1:reliable
* @return void
*/
typedef void TRRO_onControlData(void *context, const char *controller_id, const char*
msg, int len, int qos);
```

参数	含义
context	回调上下文
controller_id	控制端设备id

msg	控制消息字符串
len	字符串长度
qos	消息 qos 类型： • 0：不可靠传输，不会重传 • 1：可靠传输

远端设备

向现场设备发送数据

使用说明：该接口会向会话连接中的指定现场设备发送数据消息，仅当该远端设备拥有现场设备控制权时（ 控制授权管理 ），才可发送成功。 发送消息单条长度限制为700字节，发送频率限制为100Hz。

```
/*
* @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

注册控制消息回调

使用说明：此接口用于注册远端设备 SDK 发送的控制数据回调函数，可根据需要实现。

```
/**
* @name TRRO_registerControlDataCallback
* @brief 注册远端设备消息回调函数
* @param[in]   context          上下文指针，回调时会返回该上下文指针
* @param[in]   callback         回调函数
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerControlDataCallback(void* context,
TRRO_onControlData * callback);
```

参数	含义
context	上下文指针，回调时会返回该上下文指标用于定位
callback	回调函数

回调函数定义

```
/**
* @name TRRO_onControlData
* @brief 接收远端设备消息回调
* @param[in]    context       上下文指针，返回注册时传入的context
* @param[in]    controller_id  远端设备ID
* @param[in]    msg            消息体内容
* @param[in]    len            消息体长度
* @param[in]    qos            消息qos类型 0:unreliable, 1:reliable
* @return void
*/
typedef void TRRO_onControlData(void *context, const char *controller_id, const char*
msg, int len, int qos);
```

参数	含义
context	回调上下文，返回注册时传入的 context
controller_id	远端设备 ID
msg	控制消息字符串
len	字符串长度

qos	消息 qos 类型。 • 0：不可靠传输 • 1：可靠传输

控制授权管理

最近更新时间：2024-02-07 10:08:21

> **说明：**
> 控制授权管理用于分时管理远端设备给现场设备发送消息的权限。当有多个远端设备有权限操控某个现场设备时，可通过控制授权来限制同一时刻只有一个远端设备可以给现场设备发送控制消息，保障操控的安全性。

基本概念

权限类型

1. guest 权限

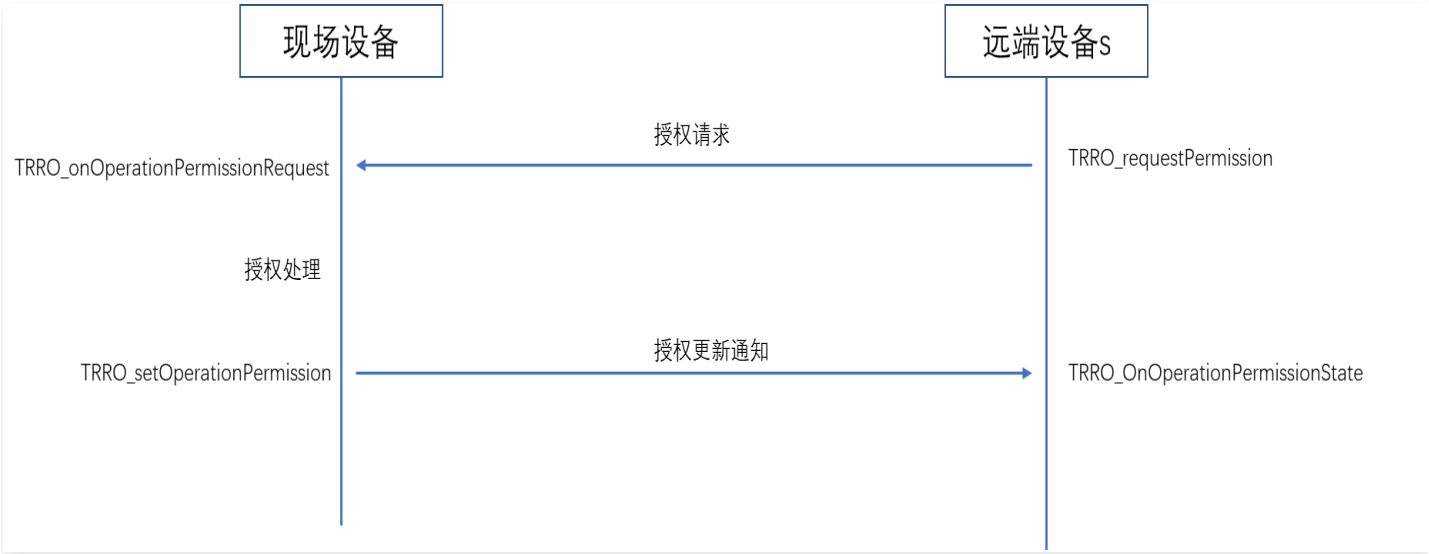
   远端设备作为旁观者，只能观看现场设备音视频内容和收到现场设备的上报数据，无法给现场设备下发控制消息。
2. master 权限

   远端设备作为操控者，既可以观看现场设备音视频内容和收到现场设备的上报数据，又可以给现场设备下发控制消息。

> **注意：**
> - 目前现场设备 SDK 限制一个现场设备同时只能给一个远端设备授权为 master 权限。
> - 当现场设备 SDK 给某个远端设备授权为 master 权限时，其他连接的远端设备会自动转为 guest 权限。

基本流程

1. 远端设备向现场设备发送授权请求请求权限， 0 为 guest 权限，1为 master 权限。
2. 现场设备根据回调的远端设备授权请求，对远端设备进行授权处理，并给相关远端设备发送授权通知。
3. 远端设备可根据现场设备的授权通知，判断当前授权状态。



授权模式

远端设备 SDK 和现场设备 SDK 可以分别配置授权模式是否为自动模式。具体配置的 json 字段为 `auto_permission` 字段，设置为1时为自动授权模式，设置为0时是应用授权模式，缺省配置默认为自动授权模式。

自动授权模式

应用可无需关心授权的处理，SDK 会自动按以下默认逻辑进行授权处理：

- 远端设备设置自动授权模式时，在发起 connect 请求时，会自动给连接的现场设备发送 master 权限授权请求。
- 现场设备设置自动授权模式时，收到远端设备的授权请求时，会自动按远端设备请求的授权权限对远端设备进行授权处理。

应用授权模式

SDK 提供相关授权接口，应用需要进行接口调用进行授权处理：

- 远端设备 SDK 不会自动发送 master 权限授权请求，需要应用显式调用权限请求接口，向现场设备请求权限授权。
- 现场设备 SDK 不会自动处理授权请求，需要应用根据回调的授权请求内容，结合应用定义的授权规则，显式调用授权接口，对远端设备进行授权。

授权接口

远端设备

注册现场设备操控权限变化通知

```
/*
*  回调注册函数
*  context 上下文指针，回调时会返回注册时传入的该指针,
*  callback 注册的 TRRO_OnOperationPermissionState 回调函数
*/

extern "C" TRRO_EXPORT void TRRO_registerOnOperationPermissionState(void* conte
xt, TRRO_OnOperationPermissionState* callback);


/*
* #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  本远端设备当前操控权限，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);
```

请求现场设备操控权限操控权限

```
/*
* @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
);
```

现场设备

注册远端设备操控权限请求通知回调

使用说明：此接口用于在现场设备接收权限控制请求，可按需选择使用。

```
/**
* @name  TRRO_registerOperationPermissionRequest
* @brief 注册远端设备操控权限请求通知回调
```

```
* @param[in]   context             上下文
* @param[in]   callback            回调函数
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerOperationPermissionRequest(void
*context, TRRO_onOperationPermissionRequest *callback);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
callback	TRRO_onOperationPermissionRequest 回调函数

```
/**
* @name TRRO_onOperationPermissionRequest
* @brief 远端设备操控权限申请通知
* @param[in]    remote_devid       请求权限的 remote deviceId
* @param[in]    permission         请求的权限，参考TrroPermission，  0: guest 只有观看权
限，1: master 完全控制权限
* @return void
*/
typedef void TRRO_onOperationPermissionRequest(void* context, const char*
remote_devid, int permission);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
remote_devid	请求权限的 remote deviceId
permission	请求的权限，参见 TrroPermission： • 0：guest 只有观看权限， • 1：master 完全控制权限

设置远端设备操作权限

使用说明：此接口用于在改现场设备设置远端设备的操作权限。

```
/**
 * @name  TRRO_setOperationPermission
```

```
 * @brief 设置远端设备操控权限，目前同时只能有一个远端设备有 master 权限，若已有远端设备是 master 权限，调用该接口设置 master权限，会自动取消之前设备的master权限然后设置新设备；
 * @param[in] remote_devid 设置权限的对端设备 ID
 * @param[in] permission 参考 TrroPermission， 0 guest，只有观看权、1 master， 完全控制权限
 * @return 1 for success, other failed
 */
extern "C" TRRO_EXPORT int TRRO_setOperationPermission(const char* remote_devid, int permission);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
callback	回调函数

会话连接诊断

最近更新时间：2024-02-20 10:59:32

> **说明：**
> 会话连接诊断，用于远端设备诊断与指定现场设备会话连接的情况。当与现场设备会话连接出现问题时，可通过该功能定位会话链路问题。

诊断流程

> **注意：**
> 诊断前，需要先对目标设备发起会话连接，然后再对目标设备进行诊断，最后从诊断回调接口中获取诊断报告。诊断时间约为10秒。

步骤1：发起设备会话连接

调用远端设备 SDK 的 connect 接口，对指定现场设备视频进行会话连接。

```
/*
* @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);
```

步骤2：注册诊断回调接口

```
/*
```

```
*  回调注册函数
*  context 上下文指针，回调时会返回注册时传入的该指针，
*  callback 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
*/

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

回调函数

```
/*
* #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);
```

步骤3：对设备会话进行诊断

调用 TRRO_connect 接口后，保持视频会话连接，通过调用诊断接口，等待从诊断回调函数中获取诊断报告。诊断时间为10秒左右。

```
/*
* @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

诊断说明

诊断输出报告为 json 格式字符串，包含现场设备和远端设备的 mqtt 服务状态、信令服务连接状态和媒体服务连接状态以及媒体链路状态等信息。

JSON 字段	含义	正常值	异常值可能原因
remote_mqtt_connect	远端设备 mqtt 状态	ok	• 与 mqtt 服务网络或端口不通 • 同名设备登录被阻塞
diag_error	诊断错误状态	ok	• 现场设备 mqtt 服务连接异常 • 远端设备 mqtt 服务连接异常
device_id	现场设备 ID	—	—
remote_mode	远端设备模式	与现场设备模式相同	• 现场设备与远端设备配置模式不一致
result	诊断详情	JSON 数组，按视频流号排列	• 诊断失败，无法获取详情
result / stream_id	现场设备视频流 ID	—	—
result / conn_fd	远端设备接收句柄 ID	—	—
result / field	现场设备详情	JSON 字符串	• 获取现场设备诊断信息失败
result / field / signal_connect	现场设备信令连接	ok	• 与信令服务网络或端口不通
result / field / media_connect	现场设备媒体连接	ok	• 媒体服务网络或端口不通 • 现场设备未推流
result / field / video_capture	现场设备视频采集	ok	• 相机采集异常 • 没有输入视频数据 • 使用外部编码流输入
result / field / video_encode	现场设备视频解码	ok	• 编码失败 • 编码器没有输入视频数据
result / field / video_transfer	现场设备视频发送	ok	• 没有进行推流 • 媒体服务或信令服务连接出错
result / field / lost	现场设备视频发送丢包	<1	• 现场设备网络发送有丢包

result / field / fps	现场设备视频发送帧率	与配置帧率相近	• 网络受限导致降帧 • cpu 或编码器处理能力不足 • 相机或输入帧率不足
result / field / bps	现场设备视频发送码率	与配置码率范围相近	• 网络受限导致发送码率不足 • 编码器没有输出或输出码率不足
result / field / rtt	现场设备网络延迟	<100ms	• 现场设备网络延迟过大
result / remote	远端设备详情	JSON字符串	—
result / remote / signal_connect	远端设备信令连接	ok	• 与信令服务网络或端口不通
result / remote / media_connect	远端设备媒体连接	ok	• 媒体服务网络或端口不通 • 现场设备未推流
result / remote / video_transfer	远端设备视频接收	ok	• 现场设备没有推流数据
result / remote / video_decode	远端设备视频解码	ok	• 没有解码数据 • 解码失败
result / remote / latency	远端设备视频延迟	30 – 150	• 视频延迟异常
result / remote / lost	远端设备视频接收丢包	<1	• 远端设备视频网络接收有丢包
result / remote / fps	远端设备视频接收帧率	与现场设备配置帧率相近	• 网络受限导致接收帧率不足 • 现场设备实际发送帧率不足
result / remote / bps	远端设备视频接收码率	与现场设备发送码率相近	• 网络受限导致接收码率不足
result / remote / rtt	远端设备网络延迟	<100ms	• 远端设备网络延迟过大

网络摄像机接入

最近更新时间：2024-02-07 10:08:21

说明：
现场设备 SDK 支持基于 RTSP 方式拉取网络相机的 h264/h265/mjpeg 流，可支持透传和转码两种相机接入方式。调用前需确认相机厂商提供的 RTSP 拉流 URL 格式，并验证 URL 对应视频流可正常播放：

- 透传模式：SDK 直接采用网络相机编码流进行传输。该模式下，SDK 无法自适应调整视频流传输码率。
- 转码模式：SDK 会先将网络相机视频流解码后再重新编码进行传输。

透传模式

视频流配置中，protocol 采用 rtsp_enc 模式，并在 cameras 配置中给出网络相机接入配置。下图给出了 config.json 文件中 streams_config 网络相机流的配置示例：

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。

```
{
 "device_id":"dev1", //修改为控制台中创建的现场设备 ID
 "device_name":"vin234",
 "device_streams":1, //如果是多路输入,修改这里的流数目，并增加 streams_config 数组中的元素个数
 "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
 "streams_config": [
      {
        "fps":25,
        "bps":2000,
        "width":1920,
        "height":1080,
        "protocol":"rtsp_enc", //网络相机透传模式
        "cameras": [
           {
              "width":1920,
              "height":1080,
              "protocol":1,
```

```
            "url":"rtsp://xxxxx" //获取厂商对应网络摄像头的拉取 url，并填写在此字段中
          }
        ]
      }
    ]
}
```

转码模式

视频流配置中，protocol 采用 normal 模式，并在 cameras 配置中给出网络相机接入配置。下图给出了 config.json 文件中 streams_config 网络相机流的配置示例：

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。

```
{
 "device_id":"dev1", //修改为控制台中创建的现场设备 ID
 "device_name":"vin234",
 "device_streams":1, //如果是多路输入,修改这里的流数目，并增加 streams_config 数组中的
元素个数
 "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
 "streams_config": [
      {
        "fps":25,
        "bps":2000,
        "width":1920,
        "height":1080,
        "protocol":"normal", //转码接入模式
        "cameras": [
          {
            "width":1920,
            "height":1080,
            "protocol":1,
            "url":"rtsp://xxxxx" //获取厂商对应网络摄像头的拉取 url，并填写在此字段中
          }
        ]
      }
    ]
}
```