物联网智能视频服务（消费版）

设备接入手册

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

设备接入手册
概述
系统模块
物模型模块
P2P 传输模块
音视频传输及对讲模块
云存储模块
自定义信令模块
AI 模块
低功耗保活模块
OTA 模块
公共数据结构
错误码
常见问题解答

设备接入手册

概述

最近更新时间：2023-07-06 10:01:11

设备端 SDK 中包含了系统、物模型、音视频传输及对讲、云存储等功能模块以及需要调用的 P2P 等三方组件。

模块功能简介

系统模块

本模块提供设备上下线通知，时间同步和认证方式获取等功能，是 IoT Video 的必选模块，需要第一个初始化。

物模型模块

设备可根据物模型中的定义上报属性、事件，并可对设备下发控制指令。本模块是 IoT Video 的必选模块，需要第二个初始化。

音视频传输及对讲模块

本模块主要实现音视频监控、音视频双向对讲功能。包含初始化音视频模块、注册监控对讲所需函数、SDK 主动发起开始停止对讲及数据接收、设备端实现监控对讲开始停止的具体操作、发送监控对讲的音视频数据、接收发起端音视频数据并进行播放；该模块是 IoT Video 的可选模块，需要在系统模块、物模型模块之后初始化。

云存储模块

本模块用于将设备端的音视频数据推送并存储在云端，回看时由观看端（例如 APP）从云端拉取数据。

自定义信令模块

本模块提供自定信令与后台服务器数据收发功能。在系统模块、物模型模块初始化完成后进行初始化，提供服务器下行数据接收回调入口，提供自定义数据向后台服务器发送入口从而以实现设备侧自定义信令与服务器的交互。

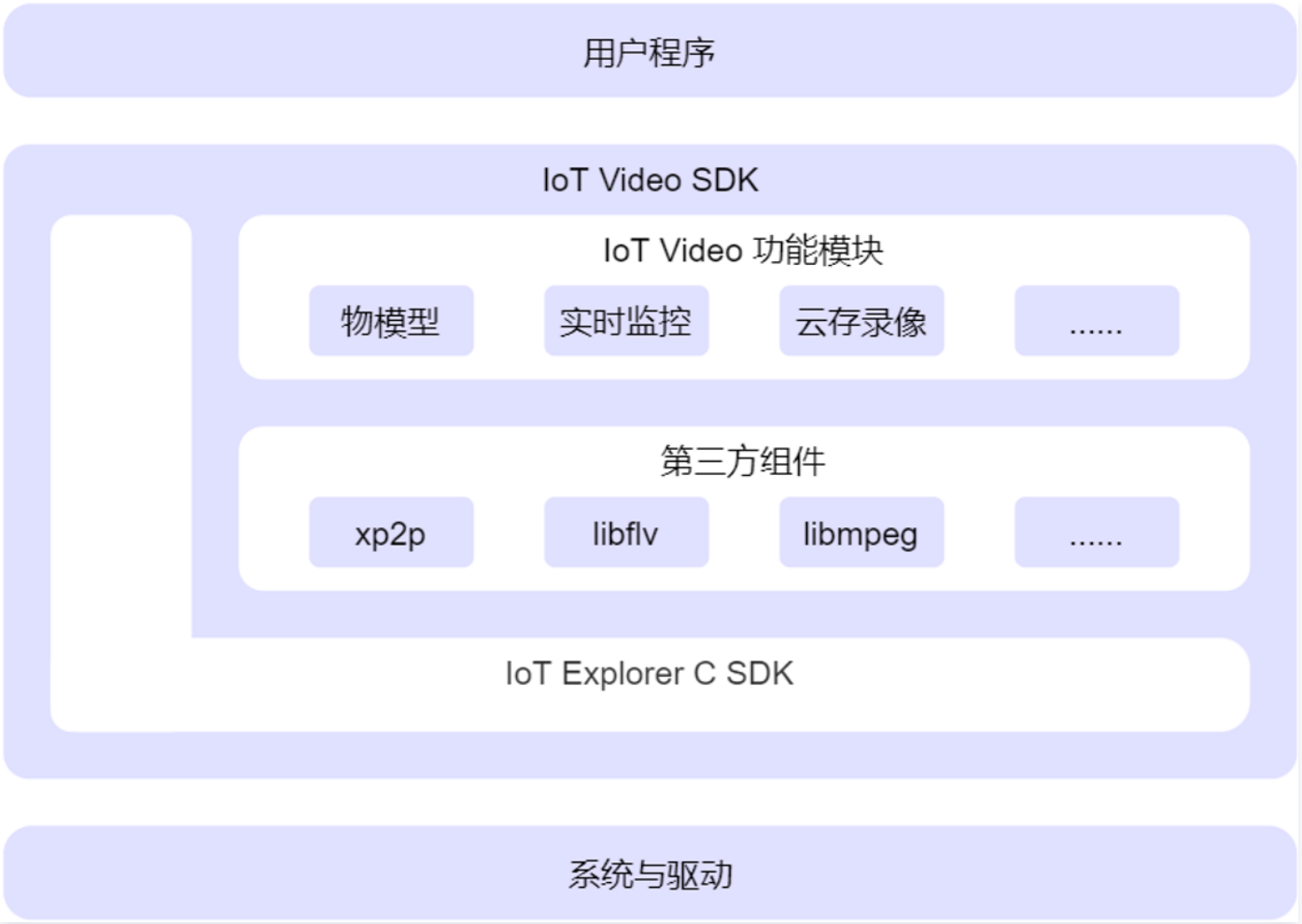
AI 模块

本模块主要实现用户将需要检测的图片上传至 COS，并请求 AI 后台完成推理，实现用户数据上云 AI 检测功能。

低功耗保活模块

低功耗设备使用低功耗保活模块使设备在主控断电或深度睡眠后自主维持保活连接，并支持远程唤醒。

SDK 框架


用户程序
IoT Video SDK
IoT Video 功能模块
物模型
实时监控
云存录像
.....
第三方组件
xp2p
libflv
libmpeg
.....
IoT Explorer C SDK
系统与驱动

系统模块

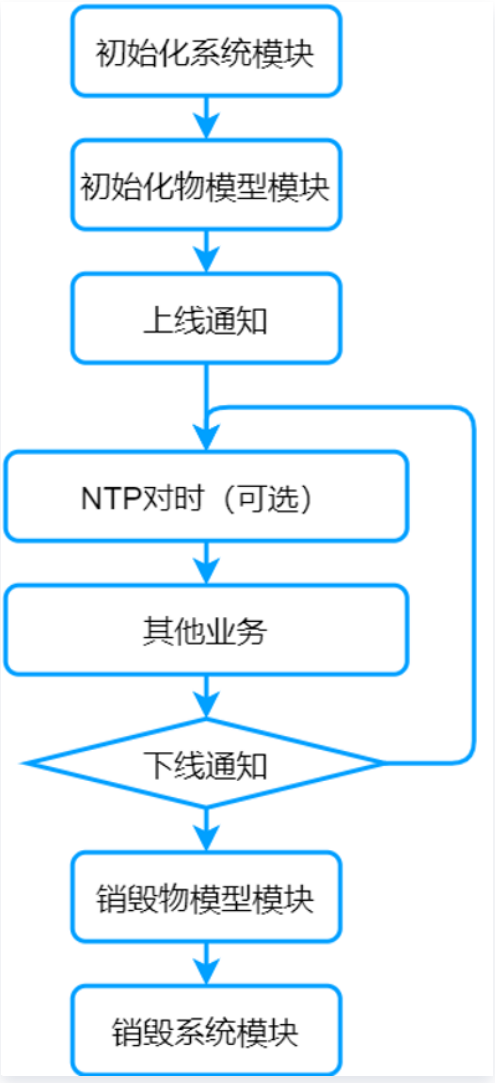
最近更新时间：2024-10-09 14:53:21

功能介绍

本模块提供设备上下线通知，时间同步和认证方式获取等功能，是 IoT Video 的必选模块，需要第一个初始化。

使用流程

模块初始化 > 上线通知 > 时间同步（可选） > 下线通知 > 模块去初始化。



接口列表

该功能模块提供以下接口：

- iv_sys_init：系统模块初始化。
- iv_sys_exit：系统模块去初始化。
- iv_sys_get_certificate_type：获取系统认证方式。

- iv_sys_get_time：获取 ntp 时间。
- iv_sys_set_log_level：设置系统打印日志。
- iv_sys_set_upload_level：设置日志上传等级。
- iv_sys_dyn_reg_device：设备动态注册。
- iv_sys_active_offline：设备主动离线接口。

用户需注册以下回调函数：

- (*iv_sys_online_cb)()：设备上线通知回调。
- (*iv_sys_offline_cb)()：设备离线通知回调。
- (*iv_func_module_status_cb)()：功能模块状态回调。

接口描述

iv_sys_init

功能描述

系统资源的初始化。包括连接参数设置、上下线回调注册等，SDK 开始时调用。

函数原型

```
int iv_sys_init(const iv_sys_init_parm_s *pstInitParm);
```

参数说明

参数名称	类型	描述	输入/输出
pstInitParm	iv_sys_init_parm_s	初始化参数。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_sys_exit

功能描述

系统资源去初始，SDK 退出时调用。

函数原型

```
int iv_sys_exit(void);
```

参数说明

无

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_sys_get_certificate_type

功能描述

获取系统的认证方式。

函数原型

```
iv_sys_certificate_type_e iv_sys_get_certificate_type(void);
```

参数说明

无

返回值

返回值	描述
IV_SYS_CERTIFICATE_TYPE_CERT	证书认证。
IV_SYS_CERTIFICATE_TYPE_KEY	密钥认证。
IV_SYS_CERTIFICATE_TYPE_BUTT	错误。

iv_sys_get_time

功能描述

设备获取 ntp 时间，单位 ms。

函数原型

```
int iv_sys_get_time(uint64_t *Param);
```

参数说明

参数名称	类型	描述	输入/输出
Param	uint64_t	ntp 时间，单位 ms。	输出

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_SYS_NTP_NOT_AVAILABLE	NTP 时间不可用，ntp 时间错误，或 ntp 时间不精确。
IV_ERR_*	失败，对应相应错误码。

iv_sys_set_log_level

功能描述

设置系统打印日志等级。

函数原型

```
void iv_sys_set_log_level(iv_sys_log_level_type_e level);
```

参数说明

参数名称	类型	描述	输入/输出
level	iv_sys_log_level_type_e	日志的打印等级。	输入

返回值

无

iv_sys_set_upload_level

功能描述

设置日志上传等级。

函数原型

```
void iv_sys_set_upload_level(iv_sys_log_level_type_e level);
```

参数说明

参数名称	类型	描述	输入/输出
level	iv_sys_log_level_type_e	日志的打印等级。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

> **注意：**
> 设置等级越低，上传的日志越多占用的带宽越大，使用前需要注意，有些版本由于资源或者其他原因，会关闭该功能，此时设置该等级会返回错误。

iv_sys_dyn_reg_device

功能描述

动态注册设备

函数原型

```
int iv_sys_dyn_reg_device(iv_sys_device_info *psys_devInfo, char *dev_psk, int dev_psk_len);
```

参数说明

参数名称	类型	描述	输入/输出
psys_devInfo	iv_sys_device_info	设备信息。	输入
dev_psk	char *	设备密钥，最大长度128。	输出
dev_psk_len	int	设备密钥长度。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

使用说明

- 视频流设备：SDK 提供两种设备认证方式，一机一密和动态注册。

- **一机一密**：控制台生成三元组信息：产品 ID（ProductId）、设备名（DeviceName）、设备密钥（DeviceSecret），在设备生产的特定环节，烧录到非易失介质中，设备 SDK 运行时读取存放的设备信息，进行设备认证。
- **动态注册**：控制台生成产品 ID（ProductId）、产品密钥（ProductSecret），在生产过程中同一产品下的所有设备在生产过程可以烧录统一的产品信息，即产品 ID（ProductId）、产品密钥（ProductSecret）。设备出厂后，第一次上电时通过动态注册的方式获取设备名和设备密钥。此接口使用psys_devInfo 传入需注册的设备信息：设备名称、产品 ID、产品密钥，其中设备名称最大长度不超过48个字符，支持数字和字母的组合，且必须保证同一产品下设备名的唯一性（例如取名为 IMEI 或者 MAC 地址）。调用成功后返回的 dev_psk 需保存在非易失存储介质中供后期使用。**此接口仅在第一次上电时调用，已注册设备重复注册无效！**

- 图片流设备：接口使用 psys_devInfo 传入需注册的设备信息，设备名称、产品 ID、产品密钥必须传入，其中设备名称最大长度不超过48个字符，支持数字和字母的组合，且同一产品下应保证设备名的唯一性。

iv_sys_active_offline

功能描述

设备主动断开与后台的连接。

函数原型

```
int iv_sys_active_offline(void);
```

参数说明

无

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

> **注意：**
> 该接口会使设备主动离线，且不会触发重新连接，针对低功耗待机等特殊场景使用。

iv_sys_online_cb

功能描述

设备上线回调，通知用户设备已经上线，并会带入网络时间参数，该回调中不要做耗时太长的操作。

网络异常时该时间参数不保证可靠，用户如需对设备进行校时，建议使用 iv_sys_get_time 接口。

函数原型

```
void (*iv_sys_online_cb)(uint64_t u64NetDateTime);
```

参数说明

参数名称	类型	描述	输入/输出
u64NetDateTime	uint64_t	网络时间，距1970年毫秒数。	输入

返回值

返回值	描述
void	无

iv_sys_offline_cb

功能描述

设备失去与 IoT Video 服务器的连接后，通过此回调通知用户，该回调中不要做耗时太长的操作。

函数原型

```
void (*iv_sys_offline_cb)(iv_sys_offline_status_type_e status);
```

参数说明

参数名称	类型	描述	输入/输出
status	iv_sys_offline_status_type_e	离线时的状态	输入

返回值

返回值	描述
void	无

iv_func_module_status_cb

功能描述

功能模块在后台服务开启状态的回调通知。

函数原型

```
void (*iv_func_module_status_cb)(uint32_t module_status);
```

参数说明

参数名称	类型	描述	输入/输出
module_status	uint32_t	功能模块状态	输入

返回值

返回值	描述
void	无

使用说明

该接口将通知各功能模块是否在后台开启服务，使用功能模块状态参数通知。

按位表示，0：未开启；1：开启；bit 0：云存模块；bit 1：Ai模块。

数据接口列表

该模块提供以下数据结构：

- iv_sys_init_parm_s：初始化参数。
- iv_sys_certificate_type_e：认证方式枚举。
- iv_sys_log_level_type_e：日志打印等级枚举。
- iv_sys_offline_status_type_e：离线状态枚举。
- iv_sys_cert_info：证书认证参数。
- iv_sys_device_info：设备三元组信息。

数据结构描述

iv_sys_init_parm_s

功能描述

系统模块初始化参数。

结构原型

```
typedef struct iv_sys_init_parm_s {
    char sys_cache_path[IV_SYS_FILE_PATH_MAX_LEN];
    char sys_store_path[IV_SYS_FILE_PATH_MAX_LEN];
    char *dev_info_path;
    iv_sys_device_info *device_info;
    uint32_t command_timeout;
```

```
    uint32_t keep_alive_ms;
    uint32_t mqtt_ping_interval_ms;
    uint8_t auto_connect_enable;
    uint32_t mqtt_recv_buf_max_size;
    uint32_t mqtt_write_buf_max_size;
    uint32_t max_channel_num;
    uint32_t comm_use_http;
    void (*iv_sys_online_cb)(uint64_t u64NetDateTime);
    void (*iv_sys_offline_cb)(iv_sys_offline_status_type_e status);
    void (*iv_func_module_status_cb)(uint32_t module_status);
} iv_sys_init_parm_s;
```

参数说明

成员名称	描述	取值
sys_cache_path	可读写目录（缓存目录），存储缓存文件，断电可以丢失。	字符串，可选。
sys_store_path	可读写目录（缓存目录），存储重要文件，断电不会丢失。	字符串，可选。
dev_info_path	device_info.json 的位置，最大长度128。	字符串，可选。
device_info	设备信息结构体。	结构体，可选。
command_timeout	连接超时时间，单位 ms。	最小值5 * 1000ms。
keep_alive_ms	后台确认掉线的超时时间，设置的越长，后台判断离线的间隔就越长，最大690 * 1000ms。	推荐 240 * 1000ms，必选。
mqtt_ping_interval_ms	设备发送 MQTT Ping 包时间间隔，设置的越小，发包就越频繁，复杂网络环境越不容易出现 mqtt 离线，必须小于 keep_alive_ms。	可选。
auto_connect_enable	自动连接使能，设备断线重连时，若失败则等待时间会翻倍，最大时间间隔60s。	• 使能：1 • 关闭：0
mqtt_recv_buf_max_size	mqtt 物模型接收 buf 大小，默认2048字节。	可选。

mqtt_write_buf_max_size	mqtt 物模型发送 buf 大小，默认2048字节。	可选。
max_channel_num	设备支持的最大通道数量，IPC 设备配置为0，NVR 或者多摄像头设备根据实际摄像头数量配置。	最大值为64。
comm_use_http	内部通信是否使用 HTTP 方式（默认使用 HTTPS 方式）。	可选， 0：使用 HTTPS 方式。 1：使用 HTTP 方式。
iv_sys_online_cb	设备上线通知。	必选参数。
iv_sys_offline_cb	设备离线通知。	必选参数。
iv_func_module_status_cb	功能模块状态（暂未使用）。	可选参数。

使用说明

1. `dev_info_path` 和 `device_info` 都是用于配置设备的三元组信息，前者指定 `device_info.json` 位置， `SDK` 会从 `device_info.json` 中读取设备三元组信息；后者是直接设置设备的三元组信息。这两个参数为互斥关系，同时设置后者生效。
2. 当设备初始化完成后，开始连接后台，当连接成功会触发 iv_sys_online_cb 通知，并传入当前 ntp 时间，网络状况较差时该时间不保证准确；当连接失败时会触发 iv_sys_offline_cb(IV_SYS_DISCONNECT_STATUS)， 此时 SDK 内部不会再进行重连，需要外部重新去初始化再进行初始化连接。
3. 当设备处于工作状态时，如果 MQTT 后台断开连接，SDK 会在大于一个 keep_alive_ms 时间后（约1.5倍的 keep_alive_ms ）感知自己离线。
4. 当 `auto_connect_enable=1` 时，设备自主感知离线后会触发 iv_sys_offline_cb(IV_SYS_RECONNECT_STATUS) 通知用户处于重连状态，设备端会自动重连后台，当自动重连成功后会触发 iv_sys_online_cb；当自动重连失败后会触发 iv_sys_offline_cb(IV_SYS_DISCONNECT_STATUS) 通知用户连接失败，需要用户重新去初始化和初始化 IoT Video SDK。
   当 `auto_connect_enable=0` 时，设备自主感知离线后会触发 iv_sys_offline_cb(IV_SYS_DISCONNECT_STATUS) 通知用户连接失败，需要用户重新去初始化和初始化 IoT Video SDK。

iv_sys_certificate_type_e

功能描述

认证方式类型枚举。

结构原型

```
typedef enum
{
    IV_SYS_CERTIFICATE_TYPE_CERT = 0,
    IV_SYS_CERTIFICATE_TYPE_KEY  = 1,

    IV_SYS_CERTIFICATE_TYPE_BUTT
} iv_sys_certificate_type_e;
```

参数说明

成员名称	描述	取值
IV_SYS_CERTIFICATE_TYPE_CERT	证书认证。	0
IV_SYS_CERTIFICATE_TYPE_KEY	密钥认证。	1
IV_SYS_CERTIFICATE_TYPE_BUTT	无效认证。	2

iv_sys_offline_status_type_e

功能描述

系统离线状态。

结构原型

```
typedef enum
{
    IV_SYS_RECONNECT_STATUS  = 0,
    IV_SYS_DISCONNECT_STATUS = 1,

    IV_SYS_INVALID_STATUS
} iv_sys_offline_status_type_e;
```

参数说明

成员名称	描述	取值
IV_SYS_RECONNECT_STATUS	重连状态，表示系统与后台连接断开，并进行重连状态。	0

IV_SYS_DISCONNECT_STATUS	离线状态，表示系统与后台连接断开，不会再重连。	1
IV_SYS_INVALID_STATUS	无效状态。	2

iv_sys_log_level_type_e

功能描述

日志打印等级枚举。

结构原型

```
typedef enum
{
    IV_eLOG_DISABLE = 0,
    IV_eLOG_ERROR   = 1,
    IV_eLOG_WARN    = 2,
    IV_eLOG_INFO    = 3,
    IV_eLOG_DEBUG   = 4
} iv_sys_log_level_type_e;
```

参数说明

成员名称	描述	取值
IV_eLOG_DISABLE	关闭日志输出。	0
IV_eLOG_ERROR	只输出错误日志。	1
IV_eLOG_WARN	输出告警和错误日志。	2
IV_eLOG_INFO	输出告警、错误和信息日志。	3
IV_eLOG_DEBUG	输出告警、错误、信息和调试日志。	4

iv_sys_cert_info

功能描述

证书认证信息。

结构原型

```
typedef struct {
    char dev_cert_file[IV_SYS_FILE_PATH_MAX_LEN];
    char dev_pkey_file[IV_SYS_FILE_PATH_MAX_LEN];
```

```
} iv_sys_cert_info;
```

参数说明

成员名称	描述	取值
dev_cert_file	设备证书位置，最大长度128。	字符串
dev_pkey_file	设备私钥位置，最大长度128。	字符串

说明：
证书认证的详细介绍参考 设备身份认证 。

iv_sys_device_info

功能描述
设备三元组信息描述结构体。
结构原型

```
typedef struct {
    char *product_id;
    char *device_name;
    union {
        iv_sys_cert_info *device_cert;
        char *device_key;
    };
} iv_sys_device_info;
```

参数说明

成员名称	描述	取值
product_id	产品 ID，最大长度10。	字符串，必选。
device_name	设备名称，最大长度48。	字符串，必选。
device_cert	设备证书认证配置。	结构体，可选。
device_key	设备密钥认证时的密钥。	字符串， 可选。

说明：

> 证书认证还是密钥认证可根据接口 `iv_sys_get_certificate_type` 判断，两种认证方式是互斥，SDK 只能固定支持一种。

示例代码

1. 系统模块初始化

```
int sys_init(void)
{
    int eErrCode = 0;

    iv_sys_init_parm_s stSysInitParameters;
    memset(&stSysInitParameters, 0, sizeof(iv_sys_init_parm_s));

    // strcpy(stSysInitParameters.sys_cache_path, "/tmp/video_cache");
    // strcpy(stSysInitParameters.sys_store_path,
"/tmp/video_storage");

#if 0
    iv_sys_device_info dev_info      = {0};
    dev_info.product_id              = "xxxxx";
    dev_info.device_name             = "xxxx";
    dev_info.device_key              = "xxxx==";
    stSysInitParameters.device_info = &dev_info;
#else
    stSysInitParameters.dev_info_path = "./device_info.json";
#endif
    stSysInitParameters.iv_sys_online_cb     = device_online;
    stSysInitParameters.iv_sys_offline_cb    = device_offline;
    stSysInitParameters.connect_timeout      = 10 * 1000;
    stSysInitParameters.keep_alive_ms        = 60 * 1000;
    stSysInitParameters.mqtt_ping_interval_ms    = 20 * 1000;
    stSysInitParameters.auto_connect_enable = 1;

    eErrCode = iv_sys_init(&stSysInitParameters);
    if (eErrCode < 0) {
        Log_e("iv_sys_init error:%d", eErrCode);
    }

    iv_sys_certificate_type_e cert_type =
iv_sys_get_certificate_type();
    Log_i("certificate type:%s", (IV_SYS_CERTIFICATE_TYPE_CERT ==
cert_type) ? "cert" : "key");
```

```
    iv_sys_set_log_level(IV_eLOG_DEBUG);

    return eErrCode;
}
```

2. 系统模块退出

```
int sys_exit(void)
```

物模型模块

最近更新时间：2024-10-09 14:53:21

该模块是 IoT Video 的必选模块，需要第二个初始化。

功能介绍

物模型部分分为：

- 属性(property)
  属性对设备能力的描述，通过对属性的修改即可实现对设备的控制,其又可分为：读写属性 (ProWritable) 和只读属性 (ProReadonly) 。
- 行为(action)
  控制设备执行特定的行为，并将执行的结果返回。行为与属性的区别，概念上行为是数据和方法的组合，行为有执行结果的返回。属性只有数据，修改属性后设备侧是否执行成功很难在属性本身体现。
- 事件(event)
  设备发生特定情况，譬如灯的开关状态发生了变化，上报事件。应用侧收到事件后按预设逻辑推送事件。

物模型由用户声明好后导出相应的 json 文件, 使用工程目录下的 `tool/codegen.py` 脚本生成代码，执行如下命令：

`./codegen.py -c <json文件>`

最终生成如下代码：

```
iv_usrex.c:用户物模型实现源文件，实现物模型初始化相关的代码，开发者不能修改这个文件
iv_usrex_func.c:用户物模型功能实现函数，实现相应 ProWritable 和 action 的回调
iv_usrex.h:用户物模型定义头文件，定义相关数据结构，开发者不能修改这个文件
```

在编译时编入工程，根据用户功能需要，在设备初始化阶段注册不同物模型处理函数，SDK 在相应模型触发时回调，用户亦可调用物模型接口主动上传物模型数据，对于设备来说，支持 ProWritable 类模型数据接收和发送, 支持 ProReadonly 和 Event 类模型数据发送， 支持 Action 模型数据接收。

使用流程

注册下发消息回调 > 物模型初始化 > 发送物模型消息 > 物模型去初始化。

接口列表

该功能模块提供以下接口：

- iv_dm_init：物模型初始化接口。
- iv_dm_exit：物模型去初始化接口。
- iv_dm_event_report：事件上报接口。
- iv_dm_property_report：属性上报接口。

- iv_dm_property_sync：属性同步接口。
- ivm_env_init：物模型环境初始化，由脚本生成。
- ivm_lock：物模型上报加锁。
- ivm_unlock：物模型上报解锁。

用户需注册以下回调函数：

- (*iv_dm_env_init_cb)()：物模型初始化参数回调。
- (*ivm_property_report_cb)()：属性上报结果回调。

接口描述

iv_dm_init

功能描述

物模型初始化，进行物模型模块资源申请，设备启动时调用。

函数原型

```
int iv_dm_init(const iv_dm_init_parm_s *pstInitParm);
```

参数说明

参数名称	类型	描述	输入/输出
pstInitParm	iv_dm_init_parm_s	初始化参数。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_dm_exit

功能描述

物模型去初始化，进行物模型模块资源释放，设备退出时调用。

函数原型

```
int iv_dm_exit(void);
```

参数说明

无

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_dm_event_report

功能描述

上传用户自定义事件消息。

函数原型

```
iv_dm_event_report(const char *event_name);
```

参数说明

参数名称	类型	描述	输入/输出
event_name	const char	事件名。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_dm_property_report

功能描述

上报属性信息。

函数原型

```
int iv_dm_property_report(const char *key, ivm_property_report_cb cb,
void *param);
```

参数说明

参数名称	类型	描述	输入/输出
key	const char *	自定义的属性 key 值。	输入

cb	ivm_property_report_cb	上报结果通知。	输入
param	void *	自定义参数。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

注意事项

- 物模型上报后会触发回调 `cb` ，告诉用户最终的上报结果。
- 必须等待当前物模型上报成功，才能进行下一次上报，否则会返回错误。
- 物模型上报的超时时间根据 `system` 模块中的 `connect_timeout` 确定。
- 尽量控制上报属性的频率，频繁的轮询上报属性会额外占用带宽。

iv_dm_property_sync

功能描述

属性同步接口。

函数原型

```
int iv_dm_property_sync(int timeout_ms)
```

参数说明

参数名称	类型	描述	输入/输出
timeout_ms	int	同步超时时间，单位：ms，建议1000。	无

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

注意事项

- 该接口是阻塞接口，只可以在单一线程中调用，禁止多线程调用。

- 调用该接口返回 `IV_ERR_NONE` 后，会将结果同步到物模型全局变量 `g_ivm_objs` 中，可以直接读取。
- 需要在设备处于上线状态时，调用该接口才能同步到正确的值，返回 `IV_ERR_DM_REQUEST_BUSY` 表示调用太频繁（SDK 限制调用频率5s一次），本次同步失败。

iv_dm_init_sync_status

功能描述

获取上线后（或重连后）SDK 内部首次同步物模型是否成功的状态。

函数原型

```
int iv_dm_init_sync_status(void)
```

参数说明

无

返回值

返回值	描述
0	同步失败，或正在同步。
1	同步成功。

注意事项

初始化时 SDK 内部会获取最新物模型，当调用本接口并返回成功后，直接读取 g_ivm_objs 即可，可以免去用户首次调用 iv_dm_property_sync 进行手动同步的操作。

ivm_env_init

功能描述

物模型环境初始化，有物模型代码脚本生成，无需修改。

函数原型

```
int ivm_env_init(void);
```

参数说明

无

返回值

返回值	描述
IV_ERR_NONE	成功。

IV_ERR_*	失败，对应相应错误码。

注意事项

该函数定义在 `iv_usrex.c` 文件中， `ivm_doi_init_ProWritable` 中的 `flag` 参数默认为0，只有当使用云API 修改物模型时， `Method` 参数配置为 `report` 时，需要将对应物模型的 `flag` 参数手动设置为1，其他情况均不用修改。

ivm_lock

功能描述

物模型互斥加锁，在上报物模型前调用。

函数原型

```
void ivm_lock(void);
```

参数说明

无

返回值

无

ivm_unlock

功能描述

物模型互斥解锁，在上报物模型后调用。

函数原型

```
void ivm_unlock(void);
```

参数说明

无

返回值

无

iv_dm_env_init_cb

功能描述

物模型初始化参数函数回调。

函数原型

```
int (*iv_dm_env_init_cb)(void);
```

参数说明

无

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

ivm_property_report_cb

功能描述

用户上报属性回调。

函数原型

```
int (*ivm_property_report_cb)(void *param, iv_dm_report_result_e
result_code);
```

参数说明

参数名称	类型	描述	输入/输出
param	void *	用户参数	输入
result_code	iv_dm_report_result_e	上报结果	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

数据结构列表

该模块提供以下数据结构：

- iv_dm_report_result_e：物模型上报结果。
- iv_dm_init_parm_s：物模型初始化参数结构体。

数据结构描述

iv_dm_report_result_e

功能描述

物模型上报结果。

结构原型

```
typedef enum
{
    REPORT_TIMEOUT  = -2,
    REPORT_REJECTED = -1,
    REPORT_ACCEPTED = 0
} iv_dm_report_result_e;
```

参数说明

成员名称	描述	取值
REPORT_TIMEOUT	物模型上报超时	-2
REPORT_REJECTED	物模型上报被拒绝	-1
REPORT_ACCEPTED	物模型上报成功	0

iv_dm_init_parm_s

功能描述

物模型初始化参数结构。

结构原型

```
typedef struct iv_dm_init_parm_s {
    int (*iv_dm_env_init_cb)(void);
} iv_dm_init_parm_s;
```

参数说明

成员名称	描述	取值
iv_dm_env_init_cb	物模型初始化参数函数回调	无

注意事项

1. 不建议频繁调用模块相关接口。
2. 初始化时 SDK 内部会获取最新物模型，直接读取 g_ivm_objs 即可，用户不必调用 iv_dm_property_sync 进行手动同步。

示例代码

1. 物模型初始化

```
    int rc = 0;
    iv_dm_init_parm_s stInitParm;
    stInitParm.iv_dm_env_init_cb = ivm_env_init;
    rc                           = iv_dm_init(&stInitParm);
    if (rc < 0) {
        Log_e("model init failed!");
        return rc;
    }
```

2. 物模型去初始化

```
iv_dm_exit();
```

3. 通过物模型使能录像

```
int iv_usrcb_ProWritable_record_enable(DeviceProperty *property)
{
    property_dbg_info(property);
    // User implementation code
    //注意：回调函数中，不能做阻塞式操作，不得做耗时的操作。会导致核心通讯线程阻
塞!!!
    iv_cm_av_data_info_s av_format;

    //设置录像音视频参数
    av_format.eAudioSampleRate      = IV_CM_AENC_SAMPLE_RATE_44100;
    av_format.eAudioMode            = IV_CM_AENC_MODE_STEREO;
    av_format.u32SampleNumPerFrame = 1024;

    av_format.eVideoType      = IV_CM_VENC_TYPE_H264;
    av_format.u32Framerate    = 30;
    av_format.u32VideoWidth   = 640;
    av_format.u32VideoHeight = 360;

    if (*(int32_t *)(property->data) == 1) {
        iv_rd_record_start(NULL, &av_format);
    } else {
        iv_rd_record_stop();
```

```
    }
    return 0;
}
```

4. 物模型上报属性

```
    ivm_lock();      //互斥加锁
    g_ivm_objs.ProReadonly.yyyy = 1;         //修改全局变量的值
    iv_dm_property_report(yyy, cb, param); //调用上报接口,注册回调
    ivm_unlock();    //互斥解锁
```

P2P 传输模块

最近更新时间：2024-10-09 14:53:21

本接口是将 `IoT Video P2P` 的接口暴露给用户，用户可以直接使用 `IoT Video P2P` 接口进行开发。

功能介绍

本文档介绍 `IoT Video P2P` 底层接口的使用方法， `IoT Video P2P` 接口定义在 `iv_av.h` 文件中，其使用与文档《 音视频传输及对讲模块说明 》中的接口使用是互斥，用户只能使用一种接口进行音视频传输。使用 `IoT Video P2P` 接口时， `IoT Video SDK` 只提供基本的数据传输能力。

接口参考

该功能模块提供以下接口：

- iv_avt_p2p_init：P2P 初始化。
- iv_avt_p2p_exit：P2P 去初始化。
- iv_avt_p2p_set_buf_watermark：设置 P2P 的拥塞控制阈值。
- iv_avt_p2p_get_send_buf：获取 P2P 的缓存数据大小。
- iv_avt_p2p_get_send_status：获取发送流的实时状态。
- iv_avt_p2p_send_command：主动发送到对端。

用户需注册以下回调函数：

- p2p_handle_send_init_cb：直播或者回放时启动的回调。
- p2p_handle_send_get_cb：获取数据回调。
- p2p_handle_send_stop_cb：直播或者回放时停止的回调。
- p2p_handle_event_notify_cb：事件通知的回调。
- p2p_handle_recv_init_cb：开始接收音频通知回调。
- p2p_handle_recv_voice_cb：接收音频数据的回调。
- p2p_handle_command_cb：命令交互的回调。

接口描述

iv_avt_p2p_init

功能描述

P2P 初始化，注册 P2P 的各种回调函数。

函数原型

```
int iv_avt_p2p_init(const iv_p2p_parm_s *pstInitParm);
```

参数说明

参数名称	类型	描述	输入/输出
pstInitParm	iv_p2p_parm_s	初始化参数。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_xx	失败，对应相应错误码。

iv_avt_p2p_exit

功能描述

P2P 初始化。本模块退出时调用，用于释放资源。

函数原型

```
int iv_avt_p2p_exit(void);
```

参数说明

无

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_xx	失败，对应相应错误码。

iv_avt_p2p_set_buf_watermark

功能描述

设置 P2P 缓存区的拥塞控制阈值。

函数原型

```
int iv_avt_p2p_set_buf_watermark(size_t low_mark, size_t warn_mark,
size_t high_mark);
```

参数说明

参数名称	类型	描述	输入/输出
low_mark	size_t	缓存数据低阈值。	输入
warn_mark	size_t	缓存数据报警阈值。	输入
high_mark	size_t	缓存数据高阈值。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_xx	失败，对应相应错误码。

说明：

设置 P2P 发送缓冲区的水位告警值(单位：字节)，用于链路拥塞感知和码率变化控制。

- 当发送缓冲区存在未发送的数据超过 warn_mark 时，会触发一次 `IV_AVT_EVENT_P2P_WATERMARK_WARN` 事件。
- 当数据超过 high_mark 时，会触发一次 `IV_AVT_EVENT_P2P_WATERMARK_HIGH` 事件，并暂停通过回调取数据。
- 直到数据低于 low_mark 时，会触发一次 `IV_AVT_EVENT_P2P_WATERMARK_LOW` ，并重新取数据。

可根据视频流码率和可用内存做动态调整，目前该设置是全局的，即对所有的视频流推送都使用同样的设置，如果三个值都设置为0，则不进行告警，用户可以通过 `iv_avt_p2p_get_send_buf` 自行做拥塞控制。

iv_avt_p2p_get_send_buf

功能描述

获取相关 `P2P` 数据流传输通道的缓冲区大小。

函数原型

```
size_t iv_avt_p2p_get_send_buf(void *handle);
```

参数说明

参数名称	类型	描述	输入/输出
handle	void *	数据传输通道句柄。	输入

返回值

返回值	描述
size_t	缓冲区大小。

> **说明：**
> 获取相关 P2P 视频流传输通道的发送缓冲区大小，用于用户自定义的 P2P 链路拥塞控制，传入由 `p2p_handle_send_init_cb` 回调创建的视频通道句柄指针 handle。

iv_avt_p2p_get_send_status

功能描述

获取相关 `P2P` 视频流传输通道的发包统计数据如发送速率。

函数原型

```
iv_p2p_send_stats_s iv_avt_p2p_get_send_status(void *handle);
```

参数说明

参数名称	类型	描述	输入/输出
handle	void *	数据传输通道句柄。	输入

返回值

返回值	描述
iv_p2p_send_stats_s	发包统计数据。

> **说明：**
> 获取相关 P2P 视频流传输通道的发包统计数据如发送速率等，用于用户自定义的 `P2P` 链路拥塞控制，传入由 `p2p_handle_send_init_cb` 回调创建的视频通道句柄指针 `handle` 。

iv_avt_p2p_send_command

功能描述

在看直播期间，设备端可以通过该接口主动发送反馈消息给对端，并可以接收对端回复。

函数原型

```
int iv_avt_p2p_send_command(const char *peer, const char *msg, size_t
msg_len,
                            unsigned char *recv_buf, size_t *recv_len,
uint32_t timeout_ms);
```

参数说明

参数名称	类型	描述	输入/输出
peer	const char *	对端的 peername。	输入
msg	char*	发送的消息内容。	输入
msg_len	size_t	发送的消息长度。	输入
recv_buf	unsigned char*	接收消息的缓存。	输出
recv_len	size_t*	接收消息的长度。	输入输出
timeout_ms	uint32_t	超时时间，单位 ms。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

说明：

- 该调用为同步阻塞方式，如果设置了较短的超时时间，则在网络繁忙或不稳定时候，该消息可能会因为超时而被丢弃。
- msg 传入的内存类型不做要求，可以是只读的，也可以是可读写的；msg 可以为任意格式字符或二进制数据，长度由 msg_len 提供，建议在16KB以内，否则会影响实时性。
- recv_buf 必须是事先分配的足够大的内存用于存放对端回复的数据，recv_len首先传入recv_buf的最大大小，接收到数据后会写入实际接收的数据大小。
- recv_buf 必须是事先分配的足够大的内存用于存放回复的数据，实际数据长度根据 recv_len 获取。
- timeout_us：命令超时时间，单位为毫秒，值为0时采用默认超时(7500ms左右)。

p2p_handle_send_init_cb

功能描述

设备端发送数据开始通知回调，入口参数为 App 端获取媒体流请求时传递的参数，可以通过“action=live”或“action=playback”来区分实时流和回放。

函数原型

```
void *(*p2p_handle_send_init_cb)(const char *params, const char *peer);
```

参数说明

参数名称	类型	描述	输入/输出
params	const char *	直播或者回放时携带的参数。	输入
peer	const char *	请求端的标识，表明对端身份。	输入

返回值

返回值	描述
handle	通道的句柄。

p2p_handle_send_get_cb

功能描述

获取设备端媒体流数据的回调，会传入 `p2p_handle_send_init_cb` 返回的句柄，该回调每10ms触发一次。

函数原型

```
iv_cm_memory_s (*p2p_handle_send_get_cb)(void *handle);
```

参数说明

参数名称	类型	描述	输入/输出
handle	void *	通道的句柄。	输入

返回值

返回值	描述
iv_cm_memory_s	返回的数据结构体，包括地址、大小和地址释放回调。

p2p_handle_send_stop_cb

功能描述

设备端发送数据停止通知回调。

函数原型

```
int (*p2p_handle_send_stop_cb)(void *handle);
```

参数说明

参数名称	类型	描述	输入/输出
handle	void *	通道的句柄。	输入

返回值

返回值	描述
int	返回数值不影响。

p2p_handle_event_notify_cb

功能描述

P2P 事件通知的回调。

函数原型

```
void (*p2p_handle_event_notify_cb)(iv_avt_event_e, void *handle);
```

参数说明

参数名称	类型	描述	输入/输出
事件	iv_avt_event_e	事件的类型。	输入
handle	void *	通道的句柄。	输入

返回值

无

使用说明

当事件类型是 `*WATERMARK*` 类型时，`handle` 是 `p2p_handle_send_init_cb` 时返回的句柄;否则，`handle` 为 `NULL` 。

p2p_handle_recv_init_cb

功能描述

P2P 开始接收 App 音频数据通知回调。

函数原型

```
void *(*p2p_handle_recv_init_cb)(const char *params, const char *peer);
```

参数说明

参数名称	类型	描述	输入/输出
params	const char*	接收音频时携带的参数。	输入
peer	const char*	对端音频标识。	输入

返回值

返回值	描述
void*	自定义句柄。

p2p_handle_recv_voice_cb

功能描述

P2P 接收 App 音频的数据回调，当 App 的发送结束时，会传入 recv_buf=NULL 且 recv_len=0。

函数原型

```
void (*p2p_handle_recv_voice_cb)(void *handle, uint8_t *recv_buf, size_t
recv_len);
```

参数说明

参数名称	类型	描述	输入/输出
handle	void*	p2p_handle_recv_init_cb 返回的句柄。	输入
recv_buf	uint8_t*	接收的音频数据首地址。	输入
recv_len	size_t	接收的音频数据大小。	输入

返回值

无

p2p_handle_command_cb

功能描述

P2P 事件通知的回调，是一个同步接口，不能有耗时太久的操作。

函数原型

```
iv_cm_memory_s (*p2p_handle_command_cb)(const char *command, size_t
cmd_len, const char *peer);
```

参数说明

参数名称	类型	描述	输入/输出
command	const char *	收到的命令。	输入
cmd_len	size_t	命令的长度。	输入
peer	const char *	发送命令端的标识。	输入

返回值

返回值	描述
iv_cm_memory_s	返回的数据。

注意事项

`iv_cm_memory_s` 中的 `buf` 地址必须是可读写的，SDK 内部会对其进行修改；否则，SDK 运行会出现错误。如果返回 `buf` 为 `NULL` 或者 `cmd_len` 为0时，表示通道断开，`APP` 侧会收到通道断开通知。

数据结构

本模块提供以下数据结构：

- iv_p2p_parm_s 音视频对讲初始化参数结构体。
- 其余数据结构请参考《 音视频传输及对讲模块说明 》。

iv_p2p_parm_s

功能描述

P2P 初始化参数结构体。

结构原型

```
typedef struct iv_avt_init_parm_s {
    void *(*p2p_handle_send_init_cb)(const char *params, const char
*peer);
    iv_cm_memory_s (*p2p_handle_send_get_cb)(void *handle);
    int (*p2p_handle_send_stop_cb)(void *handle);
    void (*p2p_handle_event_notify_cb)(iv_avt_event_e, void *handle);
    void *(*p2p_handle_recv_init_cb)(const char *params, const char
*peer);
```

```
    void (*p2p_handle_recv_voice_cb)(void *handle, uint8_t *recv_buf,
size_t recv_len);
    iv_cm_memory_s (*p2p_handle_command_cb)(const char *command, size_t
cmd_len, const char *peer);
    avt_xp2p_ops_s p2p_keep_alive;
    p2p_init_params_s p2p_init_params;
    local_net_info_s net_info;
} iv_avt_init_parm_s;
```

参数说明

成员名称	描述	取值
p2p_handle_send_init_cb	发送请求的开始回调	-
p2p_handle_send_get_cb	发送数据的回调	-
p2p_handle_send_stop_cb	发送请求的结束回调	-
p2p_handle_event_notify_cb	事件通知回调	-
p2p_handle_recv_init_cb	接收音频数据开始回调	-
p2p_handle_recv_voice_cb	发送音频数据的回调	-
p2p_handle_command_cb	信令收发的回调	-
p2p_keep_alive	P2P 保活配置	-
p2p_init_params	P2P 初始化参数	-
net_info	局域网配置（可选）	-

内存消耗统计

设备在使用 P2P 发送数据时,其内存消耗与网络带宽 bandwidth，传输的数据速率 rate 相关。

- 当 bandwidth > rate 时，正常连接时内存消耗 memory <= m * 150KB。
- 当 bandwidth <= rate 时，弱网情况下，SDK 内部缓存达到最大，其最大内存消耗 memory = (n * 512 KB+ 732 KB) * m。
  其中，m 是设备连接的 App 数量，n 是一个 App 请求的接收数据通道数量，即设备端的数据发送通道数量。

示例代码

1. P2P 初始化

```
    iv_p2p_parm_s p2p_param = {0};

    p2p_param.p2p_handle_send_init_cb        = test_flv_send_init;
    p2p_param.p2p_handle_send_get_cb         = test_flv_send_data;
    p2p_param.p2p_handle_send_stop_cb        = test_flv_send_stop;
    p2p_param.p2p_handle_event_notify_cb     = test_p2p_notify;
    p2p_param.p2p_handle_recv_init_cb        = test_recv_data_init;
    p2p_param.p2p_handle_recv_voice_cb       = test_recv_data;
    p2p_param.p2p_handle_command_cb          = test_command_handle;
    p2p_param.p2p_keep_alive.time_inter_s    = 10;
    p2p_param.p2p_keep_alive.max_attempt_num = 3;
    p2p_param.p2p_init_params.log_level      = IV_AVT_P2P_LOG_DEBUG;
    p2p_param.p2p_init_params.log_file_path  = "/tmp";
    p2p_param.p2p_init_params.log_file_size  = 1 * 1024 * 1024;

    memset(sg_p2p_chn_hanlde, 0, MAX_CHANNEL_NUM *
sizeof(p2p_chn_handle));

    int ret = iv_avt_p2p_init(&p2p_param);
    if (ret) {
        printf(" p2p init failed\n");
        return -1;
    }

#ifdef USER_CONGESTION_CTRL
    // do network congestion control by yourself
    iv_avt_p2p_set_buf_watermark(0, 0, 0);
#else
    iv_avt_p2p_set_buf_watermark(200 * 1024, 400 * 1024, 500 * 1024);
#endif
```

2. 配置拥塞控制参数

```
iv_avt_p2p_set_buf_watermark(200 * 1024, 400 * 1024, 500 * 1024);
```

3. p2p 退出

```
void p2p_sample_exit(void)
{
    iv_avt_p2p_exit();
```

```
}
```

音视频传输及对讲模块

最近更新时间：2024-08-23 17:43:51

本模块主要用于音视频监控，音视频对讲功能。

功能介绍

本模块主要实现音视频监控，音视频双向对讲功能。包含初始化音视频模块，注册监控对讲所需函数，SDK 主动发起开始停止对讲及数据接收，设备端实现监控对讲开始停止的具体操作，发送监控对讲的音视频数据，接收发起端音视频数据并进行播放；该模块是 IoT Video 的可选模块,需要在系统模块、物模型模块之后初始化。

使用流程

系统模块总流程

```
iv_avt_init //音视频传输模块初始化，注册回调
    |
（音视频传输，接收信令，接收音频）
    |
iv_avt_exit //音视频传输模块去初始化
```

音视频数据传输流程

```
iv_avt_get_av_enc_info_cb //获取音视频编码
    |
iv_avt_start_real_play_cb //开始播放
    |
while(1){iv_avt_send_stream} //推流
    |
iv_avt_stop_real_play_cb  //停止推流
    |
停止调用iv_avt_send_stream
```

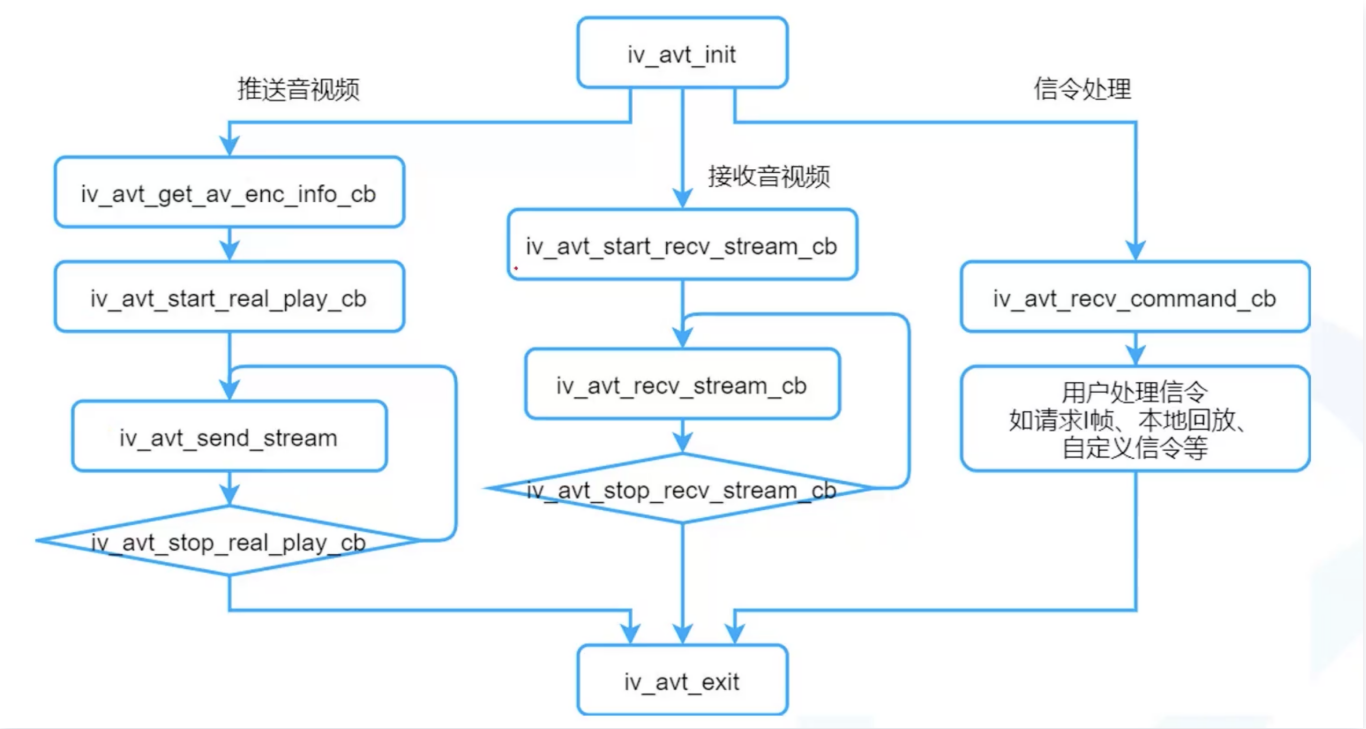
接收对向音频流程

```
iv_avt_start_recv_stream_cb //开始接收音频回调
    |
iv_avt_recv_stream_cb //接收数据流，并进行解码播放
    |
iv_avt_stop_recv_stream_cb //停止接收
```

接收信令

```
iv_avt_recv_command_cb //接收信令
    |
处理信令        //处理信令
```

基本使用流程



接口列表

该功能模块提供以下接口：

接口	说明
iv_avt_init	音视频对讲模块初始化。
iv_avt_exit	音视频对讲模块去初始化。
iv_avt_send_stream	发送对讲音视频数据。
iv_avt_get_send_stream_status	获取发送流的实时状态。
iv_avt_get_send_stream_buf	获取缓存中的实时数据量。
iv_avt_send_finish_stream	设备端结束当前点播流。
iv_avt_send_command	设备端主动发送数据到对端。

iv_avt_get_peerinfo	获取设备的 peerinfo 信息。

用户需注册以下回调函数：

回调函数	说明
iv_avt_get_av_enc_info_cb	获取音视频编码参数信息。
iv_avt_start_real_play_cb	现场音视频开始播放回调。
iv_avt_stop_real_play_cb	现场音视频停止播放回调。
iv_avt_start_recv_stream_cb	开始接收数据流回调。
iv_avt_stop_recv_stream_cb	停止接收数据流回调。
iv_avt_recv_stream_cb	接收数据回调。
iv_avt_recv_user_data_cb	监控时接收自定义数据并响应回调，已废弃。
iv_avt_notify_cb	事件通知回调。
iv_avt_recv_command_cb	接收信令回调。
iv_avt_download_file_cb	文件下载请求回调。

接口描述

iv_avt_init

功能描述

音视频对讲模块初始化。此接口在 SDK 初始化完成后调用，用于初始化音视频模块参数资源，云端发起的监控、回放处理函数，注册云端发过来的音视频数据接收函数。

函数原型

```
int iv_avt_init(const iv_avt_init_parm_s *pstInitParm);
```

参数说明

参数名称	类型	描述	输入/输出
pstInitParm	iv_avt_init_parm_s	初始化参数。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_avt_exit

功能描述

音视频对讲模块去初始化。本模块退出时调用，用于释放资源。

函数原型

```
int iv_avt_exit(void);
```

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

参数说明

无

iv_avt_send_stream

功能描述

发送对讲音视频数据。在监控请求发起时，设备端所有编码启动正常后，开始取流向请求端发送数据，注意不同清晰度视频请求时音频也需要同时发送。

函数原型

```
int iv_avt_send_stream(uint32_t visitor, uint32_t channel,
iv_avt_video_res_type_e video_res_type,
                       iv_avt_stream_type_e stream_type, void
*p_stream);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入

channel	uint32_t	通道号。	输入
video_res_type	iv_avt_video_res_type_e	视频流类型。	输入
stream_type	iv_avt_stream_type_e	音视频标识。	输入
pStream	void*	每次发送的音视频数据内容。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

使用说明

`iv_avt_start_real_play_cb` 触发后，方可调用该接口发送数据流；其中 `channel` ，`video_res_type` 参数值，需要根据 `iv_avt_start_real_play_cb` 回调参数确定。
当发送的视频流为关键帧（IDR），如有 SPS、PPS、VPS(H265) 时，需要将其与 IDR 帧放在一块内存地址中，并在 `pStream` 设置关键帧标志， `pStream` 在发送音频或视频数据时结构体类型不同，具体使用方式参考 `demo` 。

iv_avt_get_send_stream_status

功能描述

获取视频流传输通道的发包统计数据如发送速率。

函数原型

```
int iv_avt_get_send_stream_status(uint32_t visitor, uint32_t channel,
                                  iv_avt_video_res_type_e
video_res_type,
                                  iv_p2p_send_stats_s
*stream_send_status);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入

video_res_type	iv_avt_video_res_type_e	视频流类型。	输入
stream_send_status	iv_p2p_send_stats_s	音视频标识。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_avt_get_send_stream_buf

功能描述

获取 p2p 缓存中实际数据量，用于自定义拥塞控制方式使用。

函数原型

```
size_t iv_avt_get_send_stream_buf(uint32_t visitor, uint32_t channel,
                                  iv_avt_video_res_type_e
video_res_type);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入
video_res_type	iv_avt_video_res_type_e	视频流类型。	输入

返回值

返回值	描述
size_t	缓存中实时数据量的大小，单位 Byte。

iv_avt_send_finish_stream

功能描述

设备端结束当前发送的直播或者点播流。

函数原型

```
int iv_avt_send_finish_stream(uint32_t visitor, uint32_t channel,
                              iv_avt_video_res_type_e video_res_type);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入
video_res_type	iv_avt_video_res_type_e	视频流类型。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

使用说明

调用该接口设备端能够主动结束某一路视频流，可用于主动结束点播或者挂断的场景。

iv_avt_send_command

功能描述

设备端主动发送信令到对端。

函数原型

```
int iv_avt_send_command(uint32_t visitor, const char *msg, size_t
msg_len, unsigned char *recv_buf,
                        size_t *recv_len, uint32_t timeout_ms);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
msg	char*	发送的消息内容。	输入
msg_len	size_t	发送的消息长度。	输入

recv_buf	unsigned char*	接收消息的缓存。	输出
recv_len	size_t*	接收消息的长度。	输入输出
timeout_ms	uint32_t	超时时间，单位 ms。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

使用说明

该调用为同步阻塞方式，如果设置了较短的超时时间，则在网络繁忙或不稳定时候，该消息可能会因为超时而被丢弃。
msg 传入的内存类型不做要求，可以是只读的，也可以是可读写的；msg 可以为任意格式字符或二进制数据，长度由 msg_len 提供，建议在16KB以内，否则会影响实时性。
recv_buf 必须是事先分配的足够大的内存用于存放对端回复的数据，recv_len 首先传入 recv_buf 最大的大小，接收到数据后会写入实际接收的数据大小。
该接口是一个同步阻塞接口，只能在直播或者回看期间调用，如果超时时间设置为0，SDK 内部会采用默认超时（7500ms左右）。

iv_avt_get_peerinfo

功能描述

获取 peerinfo 信息。

函数原型

```
const char *iv_avt_get_peerinfo(void);
```

参数说明

无

返回值

返回值	描述
peerinfo	peerinfo 的实际信息。

> **说明：**
> 该接口为了及时获取最准确的 peerinfo，推荐 `IV_AVT_EVENT_P2P_PEER_READY` 事件回调中调用。

iv_avt_get_av_enc_info_cb

功能描述

现场音视频开始播放回调。用于观看端发起现场监控时，SDK 回调设备端以取得此次监控通路的相应音视频信息。

函数原型

```
void (*iv_avt_get_av_enc_info_cb)(uint32_t visitor, uint32_t channel,
iv_avt_video_res_type_e video_res_type,iv_cm_av_data_info_s
*av_data_info);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入
video_res_type	iv_avt_video_res_type_e	视频流类型。	输入
av_data_info	iv_cm_av_data_info_s *	音视频数据信息。	输出

返回值

无

iv_avt_start_real_play_cb

功能描述

现场音视频开始播放回调。用于观看端发起现场监控时，SDK 回调通知设备端启动相关音视频业务。

函数原型

```
void (*iv_avt_start_real_play_cb)(uint32_t visitor, uint32_t channel,
iv_avt_video_res_type_e video_res_type,void *args);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入

video_res_type	iv_avt_video_res_type_e	视频流类型。	输入
args	void *	当使用用户回放通道时，会传输些时间参数，暂未使用。	输入

返回值

无

> ⚠ **注意：**
> 当每次请求数据流时都会触发一次该回调，即使是不同的对端请求相同的 channel 和类型的数据流时，也会触发多次该回调，详细使用方法参照 `demo`； `visitor` 表示对端的 ID，表明请求者的身份；`channel` 表示摄像头的 ID，针对单摄像头设备，该值为0； `video_res_type` 表示一个摄像头传输的视频流的类型（类似 IPC 的主码流,子码流概念），分辨率不强制一一对应。

iv_avt_stop_real_play_cb

功能描述

现场音视频停止播放回调，用于观看端停止现场监控时，SDK 回调通知设备端关闭相关音视频业务。

函数原型

```
void (*iv_avt_stop_real_play_cb)(uint32_t visitor, uint32_t channel,
iv_avt_video_res_type_e video_res_type);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入
video_res_type	iv_avt_video_res_type_e	视频流类型。	输入

返回值

无

> ⚠ **注意：**
> 当每次结束数据流时都会触发一次该回调，即使是不同的对端结束相同的 channel 和类型的数据流时，也会触发多次该回调，详细使用方法参照 `demo` 。

iv_avt_start_recv_stream_cb

功能描述

通知设备开始接收对向数据流回调，并输出数据流的编码信息，当前版本支持音频。

函数原型

```
int (*iv_avt_start_recv_stream_cb)(uint32_t visitor, uint32_t channel,
iv_cm_av_data_info_s *p_av_data_info);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入
pstAvDataInfo	iv_cm_av_data_info_s *	音视频解码信息参数。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_avt_stop_recv_stream_cb

功能描述

通知设备停止接收对向数据流的回调。

函数原型

```
int (*iv_avt_stop_recv_stream_cb)(uint32_t visitor, uint32_t channel,
iv_avt_stream_type_e stream_type);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入
stream_type	iv_avt_stream_type_e	音视频标识。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_avt_recv_stream_cb

功能描述

接收对向的数据流回调。用于观看端向设备发送音视频数据时，设备端的接收数据回调，由用户实现对数据进行处理播放。

函数原型

```
int (*iv_avt_recv_stream_cb)(uint32_t visitor, uint32_t channel,
iv_avt_stream_type_e stream_type, void *p_stream);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入
stream_type	iv_avt_stream_type_e	音视频标识。	输入
pStream	void *	每次接收的音频数据内容。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_avt_recv_user_data_cb

> **说明：**
> 已废弃。

功能描述

监控过程中，由 App 发送的实时指令数据到设备，数据内容由用户自声明，可以作为控制 PTZ 命令等功能,设备端填写相应的返回值。该接口是一个同步接口，不要做延时太多的操作，已废弃，相关功能已转移至 `iv_avt_recv_command_cb` 回调的事件 `IV_AVT_COMMAND_USR_DATA` 中。

函数原型

```
void (*iv_avt_recv_user_data_cb)(uint32_t visitor,uint32_t channel,
const char *src, uint32_t src_len,
                                 iv_cm_memory_s *dst);
```

参数说明

参数名称	类型	描述	输入/输出
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入
src	char *	用户数据。	输入
src_len	uint32_t	数据长度。	输入
dst	iv_cm_memory_s	返回的数据地址和长度。	输出

返回值

无

iv_avt_notify_cb

功能描述

监控过程中，事件通知接口，用户在该回调中可以根据事件类型做相应的操作。当前版本主要是拥塞控制的事件回调。

函数原型

```
void (*iv_avt_notify_cb)(iv_avt_event_e event, uint32_t visitor,
uint32_t channel,
                         iv_avt_stream_type_e stream_type);
```

参数说明

参数名称	类型	描述	输入/输出
event	iv_avt_event_e	事件类型。	输入
visitor	uint32_t	访问者 ID。	输入

channel	uint32_t	通道号。	输入
stream_type	iv_avt_stream_type_e	音视频流数据类型。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_avt_recv_command_cb

功能描述

接收信令处理回调。

函数原型

```
int (*iv_avt_recv_command_cb)(iv_avt_command_type_e command, uint32_t
visitor, uint32_t channel,
                              iv_avt_video_res_type_e video_res_type,
void *args);
```

参数说明

参数名称	类型	描述	输入/输出
command	iv_avt_command_type_e	信令类型。	输入
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入
video_res_type	iv_avt_video_res_type_e	stream 类型。	输入
args	void *	信令参数。	输入输出

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

说明：
不同的 command，参数 args 不同，具体使用方式参照 `iot_video_demo`，回调中切勿做耗时较长的操作。

iv_avt_download_file_cb

功能描述

文件下载回调。

函数原型

```
int (*iv_avt_download_file_cb)(iv_avt_download_status_e status, uint32_t visitor,
                               uint32_t channel, void *args);
```

参数说明

参数名称	类型	描述	输入/输出
status	iv_avt_download_status_e	下载状态。	输入
visitor	uint32_t	访问者 ID。	输入
channel	uint32_t	通道号。	输入
args	void *	信令参数。	输入输出

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

说明：
不同的 status，参数 args 不同，具体使用方式参照 `iot_video_demo`，回调中切勿做耗时较长的操作。

数据结构列表

本模块提供以下数据结构：

- iv_avt_stream_type_e：音视频数据类型枚举。
- iv_avt_device_status_e：设备状态。
- iv_avt_request_type_e：请求的类型。
- iv_avt_video_res_e：视频清晰度枚举。
- iv_avt_usr_data_parm_s：用户数据信令参数。
- iv_avt_req_stream_param_s：请求流信令参数。
- iv_avt_chn_name_param_s：通道名称参数。
- iv_avt_event_e：事件类型枚举。
- iv_avt_command_type_e：信令的类型枚举。
- iv_avt_download_status_e：下载状态枚举。
- iv_avt_p2p_log_level_e p2p：日志等级。
- congestion_ctrl_s：拥塞控制信息。
- p2p_keep_alive_s p2p：保活信息。
- iv_p2p_send_stats_s p2p：发送流的状态。
- p2p_init_params_s p2p：初始化参数。
- local_net_info_s：局域网数据传输参数。
- iv_avt_init_parm_s：音视频对讲初始化参数结构体。

数据结构描述

iv_avt_stream_type_e

功能描述

音视频数据类型枚举，包括音频和视频。

结构原型

```
typedef enum
{
    IV_AVT_STREAM_TYPE_AUDIO = 0, /*audio*/
    IV_AVT_STREAM_TYPE_VIDEO = 1, /*video*/
    IV_AVT_STREAM_TYPE_AV    = 2, /*audio+video*/

    IV_AVT_STREAM_TYPE_BUTT
} iv_avt_stream_type_e;
```

参数说明

成员名称	描述	取值

IV_AVT_STREAM_TYPE_AUDIO	音视	0
IV_AVT_STREAM_TYPE_VIDEO	视频	1
IV_AVT_STREAM_TYPE_AV	音频+视频	2

iv_avt_device_status_e

功能描述

对端（例如App、小程序）请求结果。

结构原型

```
typedef enum
{
    IV_AVT_DEV_ACCEPT = 0, /* device accept app request */
    IV_AVT_DEV_REFUSE = 1, /* device refuse app request*/
    IV_AVT_DEV_STATUS_BUTT = 10
} iv_avt_device_status_e;
```

参数说明

成员名称	描述	取值
IV_AVT_DEV_ACCEPT	接受请求	0
IV_AVT_DEV_REFUSE	拒绝请求	1
IV_AVT_DEV_STATUS_BUTT	无效值	10

iv_avt_request_type_e

接口描述

请求的类型。

接口原型

```
typedef enum
{
    IV_AVT_REQUEST_SEND_STREAM = 0, /* app request device to send stream
*/
    IV_AVT_REQUEST_RECV_STREAM = 1, /* app request device to receice
stream, only support audio*/
    IV_AVT_REQUEST_BUTT
} iv_avt_request_type_e;
```

参数说明

成员名称	描述	取值
IV_AVT_REQUEST_SEND_STREAM	请求设备发送数据流。	0
IV_AVT_REQUEST_RECV_STREAM	请求设备接收数据流。	1
IV_AVT_REQUEST_BUTT	无效值。	10

使用说明

用于事件 `IV_AVT_COMMAND_REQ_STREAM` 。

iv_avt_video_res_type_e

功能描述

视频流类型参数，用于区分视频流的类型，对具体对应的视频的分辨率用户自行决定，没有严格要求。

结构原型

```
typedef enum
{
    IV_AVT_VIDEO_RES_FL = 0,   // fluency stream
    IV_AVT_VIDEO_RES_SD = 1,   // standard stream
    IV_AVT_VIDEO_RES_HD = 2,   // high stream
    IV_AVT_VIDEO_RES_PB = 3,   // playback stream

    IV_AVT_VIDEO_RES_BUTT
} iv_avt_video_res_type_e;
```

参数说明

成员名称	描述	取值
IV_AVT_VIDEO_RES_FL	流畅流	0
IV_AVT_VIDEO_RES_SD	标清流	1
IV_AVT_VIDEO_RES_HD	高清流	2
IV_AVT_VIDEO_RES_PB	回放流	3
IV_AVT_VIDEO_RES_BUTT	无效流	4

iv_avt_usr_data_parm_s

功能描述

用户数据信令的实际参数。

结构原型

```
typedef struct {
    char *src;            // input data
    uint32_t src_len;     // input data length
    iv_cm_memory_s dst;   // output data
} iv_avt_usr_data_parm_s;
```

参数说明

成员名称	描述	取值
src	信令输入的数据	-
src_len	参数的长度	-
dst	信令返回的数据	-

注意事项

返回的数据地址 `dst->buf` 需要用户保证当前回调函数退出后，其生命周期仍然有效。为了避免内存泄露，用户不需要释放该地址，可以通过注册 `dst->buf_free_fn` 回调， `IoT Video SDK` 会在合适的时机调用 `dst->buf_free_fn` 释放该地址，如果该地址不需要释放，则将 `dst->buf_free_fn` 置为 `NULL` 。用户返回的 `dst->buf` 地址必须是可读写的，SDK 内部可能会修改；如果返回 `buf` 为 `NULL` 或者 `cmd_len` 为0时，表示通道断开，`App` 侧会收到通道断开通知。

iv_avt_req_stream_param_s

功能描述

`IV_AVT_COMMAND_REQ_STREAM` 事件对应的参数。

结构原型

```
typedef struct {
    iv_avt_request_type_e request_type;      // input data
    iv_avt_device_status_e request_result;   // output data
} iv_avt_req_stream_param_s;
```

参数说明

成员名称	描述	取值

request_type	请求的数据类型	输入数据。
request_result	请求结果	输出数据，用户决定是否响应该请求。

iv_avt_chn_name_param_s

功能描述

`IV_AVT_COMMAND_CHN_NAME` 事件对应的参数。

结构原型

```
typedef struct {
    iv_cm_memory_s name;
    uint8_t is_online;
} iv_avt_chn_name_param_s;
```

参数说明

成员名称	描述	取值
name	通道名称	输入数据。
is_online	是否在线	• 0：离线。 • 1：在线。

注意事项

返回的数据地址 `name->buf` 需要用户保证当前回调函数退出后，其生命周期仍然有效。为了避免内存泄露，用户不需要释放该地址，可以通过注册 `name->buf_free_fn` 回调， `IoT Video SDK` 会在合适的时机调用 `dst->buf_free_fn` 释放该地址，如果该地址不需要释放，则将 `name->buf_free_fn` 置为 `NULL` 。

用户返回的 `name->buf` 地址必须是可读写的，SDK 内部可能会修改；如果返回 `buf` 为 `NULL` 或者 `cmd_len` 为0时，表示通道断开， `App` 侧会收到通道断开通知。

`name` 会被嵌入到 json 格式中，不能有不满足 json 的特殊字符，推荐使用大小写字母和数字的组合，避免使用特殊字符。

iv_avt_event_e

功能描述

事件类型。

结构原型

```
typedef enum
{
    IV_AVT_EVENT_P2P_PEER_READY         = 0,
```

```
    IV_AVT_EVENT_P2P_PEER_CONNECT_FAIL = 1,
    IV_AVT_EVENT_P2P_PEER_ERROR        = 2,
    IV_AVT_EVENT_P2P_PEER_ADDR_CHANGED = 3,
    IV_AVT_EVENT_P2P_WATERMARK_LOW     = 4,
    IV_AVT_EVENT_P2P_WATERMARK_WARN    = 5,
    IV_AVT_EVENT_P2P_WATERMARK_HIGH    = 6,
    IV_AVT_EVENT_P2P_LOCAL_NET_READY   = 7,
    IV_AVT_EVENT_P2P_BUTT
} iv_avt_event_e;
```

参数说明

成员名称	描述	取值
IV_AVT_EVENT_P2P_PEER_READY	P2P 初始化完成通知。	0
IV_AVT_EVENT_P2P_PEER_CONNECT_FAIL	P2P 连接 STUN 服务器失败。	1
IV_AVT_EVENT_P2P_PEER_ERROR	检测网络错误。	2
IV_AVT_EVENT_P2P_PEER_ADDR_CHANGED	P2P 地址方式变化。	3
IV_AVT_EVENT_P2P_WATERMARK_LOW	P2P 缓存数据低于最低值通知。	4
IV_AVT_EVENT_P2P_WATERMARK_WARN	P2P 缓存数据超过报警值通知。	5
IV_AVT_EVENT_P2P_WATERMARK_HIGH	P2P 缓存数据超过最大值通知。	6
IV_AVT_EVENT_P2P_LOCAL_NET_READY	P2P 局域网完成。	7
IV_AVT_EVENT_P2P_BUTT	无效事件。	8

注意事项

出现 `IV_AVT_EVENT_P2P_PEER_ADDR_CHANGED` 事件，SDK 内部会进行重新协商，但此时 P2P 通道已断开，需要停止送入数据。

出现 `IV_AVT_EVENT_P2P_PEER_ERROR` 或 `IV_AVT_EVENT_P2P_PEER_CONNECT_FAIL` 事件时，如果用户只需要局域网模式，就不用处理这两个事件，如果用户需要广域网通信，就需要用户手动重新初始化 SDK。

如果用户主动修改设备端 `IP` 地址，P2P 通道可能恢复较慢，受 `MQTT` 保活时间影响，可以在 `IP` 地址修改后直接重新初始化所有接口来加快 `P2P` 通道恢复时间。

iv_avt_command_type_e

功能描述

信令的类型。

结构原型

```
typedef enum
{
    IV_AVT_COMMAND_USR_DATA    = 0,  // usr define data, for example ptz
    IV_AVT_COMMAND_REQ_STREAM  = 1,  // request stream (send or receive)
    IV_AVT_COMMAND_CHN_NAME    = 2,  // get channel name
    IV_AVT_COMMAND_REQ_IFRAME  = 3,  // request I frame

    // playback command
    IV_AVT_COMMAND_PLAYBACK_PAUSE,         // pause playback
    IV_AVT_COMMAND_PLAYBACK_RESUME,        // resume playback
    IV_AVT_COMMAND_PLAYBACK_QUERY_MONTH,   // query record info of month
    IV_AVT_COMMAND_PLAYBACK_QUERY_DAY,     // query record info of day
    IV_AVT_COMMAND_PLAYBACK_SEEK,          // seek record position
    IV_AVT_COMMAND_PLAYBACK_FF,            // playback fast forward(快进)
    IV_AVT_COMMAND_PLAYBACK_SPEED,         // playback speed(快放 or 慢放)
    IV_AVT_COMMAND_PLAYBACK_REWIND,        // rewind record(倒放)

    IV_AVT_COMMAND_TYPE_BUTT
} iv_avt_command_type_e;
```

参数说明

成员名称	描述	取值
IV_AVT_COMMAND_USR_DATA	用户数据信令	0
IV_AVT_COMMAND_REQ_STREAM	请求流信令	1
IV_AVT_COMMAND_CHN_NAME	获取通道名称信令	2
IV_AVT_COMMAND_REQ_IFRAME	请求I帧信令	3
IV_AVT_COMMAND_PLAYBACK_PAUSE	暂停回放	4
IV_AVT_COMMAND_PLAYBACK_RESUME	继续回放	5
IV_AVT_COMMAND_PLAYBACK_QUERY_MONTH	按月查询录像	6
IV_AVT_COMMAND_PLAYBACK_QUERY_DAY	按天查询录像	7
IV_AVT_COMMAND_PLAYBACK_SEEK	移动播放位置	8
IV_AVT_COMMAND_PLAYBACK_FF	快进	9

IV_AVT_COMMAND_PLAYBACK_SPEED	设置播放速度	10
IV_AVT_COMMAND_PLAYBACK_REWIND	倒放	11
IV_AVT_COMMAND_TYPE_BUTT	信令的数量	-

注意事项

`IV_AVT_COMMAND_CHN_NAME` 仅在多摄像头设备时触发，根据 `system` 模块初始化时参数 `max_channel_num` 决定触发次数，每一次触发设置一个通道的名称，如果其中一次该信令的返回值不为 IV_ERR_NONE，则会停止后续触发，只有绑定了名称的 `channel` 才能正常工作。

iv_avt_download_status_e

功能描述

视频下载状态枚举，一般用于 App（或小程序）下载设备内的本地录像。

结构原型

```
typedef enum
{
    IV_AVT_DOWNLOAD_STATUS_START,  // start to download
    IV_AVT_DOWNLOAD_STATUS_RUNNING,
    IV_AVT_DOWNLOAD_STATUS_STOP,  // stop to download
    IV_AVT_DOWNLOAD_STATUS_BUTT
} iv_avt_download_status_e;
```

参数说明

成员名称	描述	取值
IV_AVT_DOWNLOAD_STATUS_START	开始下载	0
IV_AVT_DOWNLOAD_STATUS_RUNNING	下载中	1
IV_AVT_DOWNLOAD_STATUS_STOP	停止下载	2
IV_AVT_DOWNLOAD_STATUS_BUTT	无效状态	-

iv_avt_p2p_log_level_e

功能描述

日志等级。

结构原型

```
typedef enum
```

```
{
    IV_AVT_P2P_LOG_DISABLE = 0,
    IV_AVT_P2P_LOG_ERROR   = 1,
    IV_AVT_P2P_LOG_WARN    = 2,
    IV_AVT_P2P_LOG_INFO    = 3,
    IV_AVT_P2P_LOG_DEBUG   = 4,
    IV_AVT_P2P_LOG_VERBOSE = 5
} iv_avt_p2p_log_level_e;
```

参数说明

成员名称	描述	取值
IV_AVT_P2P_LOG_DISABLE	关闭 p2p 日志。	0
IV_AVT_P2P_LOG_ERROR	只打印 p2p 的 error 类型。	1
IV_AVT_P2P_LOG_WARN	打印 p2p 的 error、 warning 类型。	2
IV_AVT_P2P_LOG_INFO	打印 p2p 的 error、warning、info 类型。	3
IV_AVT_P2P_LOG_DEBUG	打印 p2p 的 error、warning、info、debug。	4
IV_AVT_P2P_LOG_VERBOSE	打印 p2p 的所有日志。	5

iv_avt_p2p_nic_type_e

功能描述

网卡类型。

结构原型

```
typedef enum
{
    IV_AVT_P2P_NIC_NORMAL = 0,
    IV_AVT_P2P_NIC_NAT_EX = 1,
} iv_avt_p2p_nic_type_e;
```

参数说明

成员名称	描述	取值
IV_AVT_P2P_NIC_NORMAL	正常网卡。	0

IV_AVT_P2P_NIC_NAT_EX	外挂 NAT 网卡。	1

iv_avt_p2p_tp_type_e

功能描述

P2P 网络传输方式。

结构原型

```
typedef enum
{
    IV_AVT_P2P_UDP = 0,
    IV_AVT_P2P_TCP = 1,
} iv_avt_p2p_tp_type_e;
```

参数说明

成员名称	描述	取值
IV_AVT_P2P_UDP	P2P 使用 UDP 协议。	0
IV_AVT_P2P_TCP	P2P 使用 TCP 协议。	1

congestion_ctrl_s

功能描述

拥塞控制参数。

结构原型

```
typedef struct {
    size_t low_mark;
    size_t warn_mark;
    size_t high_mark;
    bool enable;
} congestion_ctrl_s;
```

参数说明

成员名称	描述	取值
low_mark	缓存数据最低值，单位 Byte。	-
warn_mark	缓存数据告警值，单位 Byte。	-

high_mark	缓存数据最高值，单位 Byte。	-
enable	拥塞控制使能。	-

使用说明

当 `enable=0` 时，无拥塞控制事件通知；当 `enable=1` 时，用户需要设置 `low_mark`，`warn_mark`，`high_mark` 值，`IoT Video SDK` 会根据网络环境和用户设置参数，触发相应的事件通知。

- 当 `IoT Video SDK` 内部缓存的数据超过 `warn_mark` 时，会触发一次 `IV_AVT_EVENT_P2P_WATERMARK_WARN` 事件通知，表明缓存较多了，用户需要降低数据流的发送。
- 当 `IoT Video SDK` 内部缓存的数据超过 `high_mark` 时，会触发一次 `IV_AVT_EVENT_P2P_WATERMARK_HIGH` 事件通知，表明内存缓存太多，网络环境太差，用户需要降低或停止数据流的发送。
- 当 `IoT Video SDK` 内部缓存的数据恢复到 `low_mark` 时，会触发一次 `IV_AVT_EVENT_P2P_WATERMARK_LOW` 事件通知，表明内存缓存正常，用户可以正常发送数据了。

p2p_keep_alive_s

功能描述

p2p 保活参数。

结构原型

```
typedef struct p2p_keep_alive_s {
    uint8_t time_inter_s;
    uint8_t max_attempt_num;
} p2p_keep_alive_s;
```

参数说明

成员名称	描述	取值
time_inter_s	P2P 保活最大时间间隔，单位 s。	0表示关闭 p2p 保活功能。
max_attempt_num	P2P 保活最大尝试次数，最大10。	推荐3，0表示关闭 p2p 保活功能。

注意事项

p2p 保活通过与服务器之间进行通信，判断设备端网络状态是否发生变化，特别是设备端网络出现中断后再连接时，引起设备所在网络的拓扑结构的变化，如果网络拓扑发生变化，IoT Video SDK 内部会重新进行 p2p 协商，从而保证 p2p 通信正常。

`time_inter_s` 表示设备与服务器之间每次通信的最大时间间隔，设置的太大，设备网络断开到恢复的时间太短检测不到，可能会导致 p2p 通信失败，设置的太小，通信过于频繁占用资源较多，推荐设置5s到10s。

`max_attempt_num` 表示设备每次与服务器通信探测包数量，设置的太大，探测包太多影响设备的网络资源，设置

的太少，网络差的情况下，判断不准确，推荐设置3。
`time_inter_s` 和 `max_attempt_num` 其中一个为0时，表示关闭 p2p 保活机制，在这种情况下 IoT Video SDK 内部会通过 system 模块中的 `keep_alive_ms` (MQTT 保活时间)进行网络判断，该值较大，灵敏度不如 p2p 保活参数。

iv_p2p_send_stats_s

接口描述
P2P 数据传输的状态。
结构原型

```
typedef struct {
    uint32_t inst_net_rate;
    uint32_t ave_sent_rate;
    uint64_t sum_sent_acked;
    uint32_t link_mode;
} iv_p2p_send_stats_s;
```

参数说明

成员名称	描述	取值
inst_net_rate	瞬时发送速率，随网速变化会有较大波动。	字节/秒
ave_sent_rate	过去一秒内累计发送并得到对端确认的数据，即平均发送速率。	字节/秒
sum_sent_acked	累计发送并得到对端确认的数据。	字节
link_mode	当前 P2P 链路的连接模式：0 无效；1 直连；2 转发。	整数

p2p_init_params

接口描述
p2p 初始化时配置参数。

```
typedef struct {
    iv_avt_p2p_log_level_e log_level;
    const char *log_file_path;
    uint32_t log_file_size;
    uint32_t mtu_size;
    iv_avt_p2p_nic_type_e nic_type;
    uint32_t sender_interval_ms;
    uint32_t keepalive_interval_s;
```

```
    uint32_t access_retry_times;
    iv_avt_p2p_tp_type_e protocol;
} p2p_init_params_s;
```

参数说明

成员名称	描述	取值
log_level	P2P 模块日志级别。	iv_avt_p2p_log_level_e
log_file_path	P2P 模块日志文件路径，若为 NULL 则输出到 stderr 或串口终端，建议配置在"/tmp"路径。	char*
log_file_size	P2P 模块日志文件大小限制，实际占用空间最大为该值两倍，SDK 会自动回滚。	uint32_t
mtu_size	P2P 模块 UDP MTU 大小。	范围 500-1460，若为0则采用默认值1460。
nic_type	设备采用某些特殊的 NAT 网卡时需要进行设置。	一般置为0。
sender_interval_ms	发送数据的 timer 间隔，默认为10毫秒，在码率较低且对功耗要求较高时可以适当改大。	一般置为0即为默认值。
keepalive_interval_s	P2P 与 STUN 服务器的保活间隔，默认为10秒。	一般置为0即为默认值。
access_retry_times	连接 STUN 服务器的重试次数，默认为10，实际的重试次数为该值的3倍。	一般置为0即为默认值。
protocol	一般置为0，当设备处于不支持 UDP 包的网络环境时，可以设置该参数为IV_AVT_P2P_TCP。	一般置为0即为默认值。

local_net_info_s

功能描述

局域网传输配置参数。

结构原型

```
typedef struct {
    int probe_port;         //局域网探测的端口
    int trans_port;         //局域网数据传输的端口
    char local_addr[64];    //设备在局域网的IP地址
    char *vendor_id;        //厂商ID,用于区分不同厂商的设备
```

```
    char *device_id;        //设备ID,用于区分设备
} local_net_info_s;
```

参数说明

成员名称	描述	取值
probe_port	探测监听端口，用于接收广播消息，设置为0表示关闭局域网功能。	int
trans_port	局域网数据传输端口号，设置为0表示关闭局域网功能。	int
local_addr	局域网地址。	char
vendor_id	厂商 ID。	char*
device_id	设备 ID。	char*

注意事项

- `probe_port` 和 `trans_port` 任意值为0，都表示关闭局域网通信功能。
- `local_addr` 设置当前设备的局域网地址，如 `192.168.0.22` 等。
- `vendor_id` 表示厂商 ID，用于区分不同厂商,如果设置为 NULL，SDK 内部使用三元组信息中的 `ProductId` 替代，同一类设备的 `vendor_id` 相同，只能由大小写字母、数字、下划线组成，不能有特殊字符。
- `device_id` 表示设备 ID，用于区分不同设备，如果设置为 NULL，SDK 内部使用三元组信息中的 `DeviceName` 替代，同一类设备的 `device_id` 要保证唯一性，只能由大小写字母、数字、下划线组成，不能有特殊字符。

iv_avt_init_parm_s

功能描述

音视频对讲初始化参数结构体。

结构原型

```
typedef struct iv_avt_init_parm_s {
    uint32_t max_frame_size;
    uint32_t max_connect_num;
    void (*iv_avt_get_av_enc_info_cb)(uint32_t visitor, uint32_t
channel,
                                      iv_avt_video_res_type_e
video_res_type,
                                      iv_cm_av_data_info_s
*av_data_info);
```

```
    void (*iv_avt_start_real_play_cb)(uint32_t visitor, uint32_t
channel,
                                      iv_avt_video_res_type_e
video_res_type, void *args);
    void (*iv_avt_stop_real_play_cb)(uint32_t visitor, uint32_t channel,
                                     iv_avt_video_res_type_e
video_res_type);
    int (*iv_avt_start_recv_stream_cb)(uint32_t visitor, uint32_t
channel,
                                       iv_cm_av_data_info_s
*p_av_data_info);
    int (*iv_avt_stop_recv_stream_cb)(uint32_t visitor, uint32_t
channel,
                                      iv_avt_stream_type_e stream_type);
    int (*iv_avt_recv_stream_cb)(uint32_t visitor, uint32_t channel,
                                 iv_avt_stream_type_e stream_type, void
*p_stream);

    /*已废弃，通过IV_AVT_COMMAND_USR_DATA实现*/
    void (*iv_avt_recv_user_data_cb)(uint32_t visitor, uint32_t channel,
const char *src,
                                     uint32_t src_len, iv_cm_memory_s
*dst);
    void (*iv_avt_notify_cb)(iv_avt_event_e event, uint32_t visitor,
uint32_t channel,
                             iv_avt_video_res_type_e video_res_type);
    int (*iv_avt_recv_command_cb)(iv_avt_command_type_e command,
uint32_t visitor, uint32_t channel,
                                  iv_avt_video_res_type_e
video_res_type, void *args);
    int (*iv_avt_download_file_cb)(iv_avt_download_status_e status,
uint32_t visitor,
                                   uint32_t channel, void *args);
    congestion_ctrl_s congestion;
    p2p_keep_alive_s p2p_keep_alive;
    p2p_init_params_s p2p_init_params;
    local_net_info_s net_info;
} iv_avt_init_parm_s;
```

参数说明

成员名称	描述	取值

max_frame_size	发送数据的最大值，发送音视频或自定义数据时，保证每次数据的大小小于此值。	单位 KB
max_connect_num	当前设备支持的最大连接数，即连接的 App 数量。	最大 32
iv_avt_get_av_enc_info_cb	获取设备编码信息回调。	-
iv_avt_start_real_play_cb	现场监控开启回调。	-
iv_avt_stop_real_play_cb	现场监控停止回调。	-
iv_avt_start_recv_stream_cb	通知设备对音频开始的回调。	-
iv_avt_stop_recv_stream_cb	通知设备对音频结束的回调。	-
iv_avt_recv_stream_cb	接收观看端发来的音视频数据回调。	-
iv_avt_recv_user_data_cb	已废弃。	-
iv_avt_notify_cb	事件通知回调。	-
iv_avt_recv_command_cb	接收信令通知回调。	-
iv_avt_download_file_cb	下载本地录像通知回调。	-
congestion	拥塞控制配置。	-
p2p_keep_alive	P2P 保活配置。	-
p2p_init_params	P2P 初始化参数。	-
net_info	局域网配置（可选）。	-

注意：
所有回调切勿做耗时较长的操作。

注意事项

内存消耗统计方法

内存开销公式如下：

```
memory(KByte) =  n * 2 * max_frame_size(KB) + 120KB
```

其中，n 是通道数量，max_frame_size 是设置的最大帧的大小。

示例代码

1. 对讲模块初始化

```
int av_talk_init(void)
{
        int ret = 0;

    iv_avt_init_parm_s stAvtInitParameters;
    memset(&stAvtInitParameters, 0, sizeof(iv_avt_init_parm_s));
    stAvtInitParameters.max_frame_size    = 384;              // 384kB
    stAvtInitParameters.max_connect_num   = MAX_CONNECT_NUM;  //
    stAvtInitParameters.congestion.enable = true;
#ifdef USER_CONGESTION_CTRL
    stAvtInitParameters.congestion.low_mark  = 0;
    stAvtInitParameters.congestion.warn_mark = 0;
    stAvtInitParameters.congestion.high_mark = 0;
#else
    stAvtInitParameters.congestion.low_mark  = 200 * 1024;
    stAvtInitParameters.congestion.warn_mark = 400 * 1024;
    stAvtInitParameters.congestion.high_mark = 500 * 1024;
#endif

    stAvtInitParameters.p2p_keep_alive.time_inter_s    = 10;
    stAvtInitParameters.p2p_keep_alive.max_attempt_num = 4;
    stAvtInitParameters.iv_avt_get_av_enc_info_cb      =
av_talk_get_enc_info;
    stAvtInitParameters.iv_avt_start_real_play_cb      =
av_talk_start_real_play;
    stAvtInitParameters.iv_avt_stop_real_play_cb       =
av_talk_stop_real_play;
    stAvtInitParameters.iv_avt_start_recv_stream_cb    =
av_talk_start_recv_stream;
```

```
    stAvtInitParameters.iv_avt_recv_stream_cb          =
av_talk_recv_stream;
    stAvtInitParameters.iv_avt_stop_recv_stream_cb     =
av_talk_stop_recv_stream;
    stAvtInitParameters.iv_avt_notify_cb               =
av_talk_notify_process;
    stAvtInitParameters.iv_avt_recv_command_cb         =
av_talk_command_proc;
    stAvtInitParameters.p2p_init_params.log_level      =
IV_AVT_P2P_LOG_DEBUG;
    stAvtInitParameters.p2p_init_params.log_file_path  = "/tmp";
    stAvtInitParameters.p2p_init_params.log_file_size  = 1 * 1024 *
1024;

    ret = iv_avt_init(&stAvtInitParameters);
    if (ret < 0) {
        Log_e("iv_avt_init error:%d", ret);
        return ret;
    }
    return ret;
}
```

2. 发送音视频数据

当 `iv_avt_start_real_play_cb;` 回调触发后，可调用如下接口发送音视频数据：

```
rc = iv_avt_send_stream(visitor, p_enc_handle->channel, p_enc_handle-
>video_res_type,
                        IV_AVT_STREAM_TYPE_VIDEO, &v_stream);
rc = iv_avt_send_stream(visitor, p_enc_handle->channel, p_enc_handle-
>video_res_type,
                        IV_AVT_STREAM_TYPE_AUDIO, &a_stream);
```

3. 接收音视频数据

```
int av_talk_recv_stream(uint32_t visitor, uint32_t channel,
iv_avt_stream_type_e stream_type,
                        void *pStream)
{
    return qcloud_av_dec_play(visitor, channel, stream_type, pStream);
}
```

4. 自定义数据收发

```
int av_talk_command_proc(iv_avt_command_type_e command, uint32_t
visitor, uint32_t channel,
                         iv_avt_video_res_type_e video_res_type, void
*args)
{
    Log_d("command %d visitor %d channel %d stream %d args %p", command,
visitor, channel,
          video_res_type, args);
    int rc = 0;

    switch (command) {
        case IV_AVT_COMMAND_USR_DATA: {
            iv_avt_usr_data_parm_s *usr_data = (iv_avt_usr_data_parm_s
*)args;
            av_talk_recv_user_data(visitor, channel, usr_data->src,
usr_data->src_len,
                                   &usr_data->dst);
            break;
        }

        case IV_AVT_COMMAND_REQ_STREAM: {
            iv_avt_req_stream_param_s *req_param =
(iv_avt_req_stream_param_s *)args;
            Log_d("type %d\n", req_param->request_type);
            req_param->request_result = IV_AVT_DEV_ACCEPT;
            break;
        }

        case IV_AVT_COMMAND_CHN_NAME: {
            iv_avt_chn_name_param_s *chn_name = (iv_avt_chn_name_param_s
*)args;
            rc = av_talk_get_dev_name_proc(visitor, channel, &chn_name-
>name, &chn_name->is_online);
            break;
        }

        case IV_AVT_COMMAND_REQ_IFRAME: {
            Log_i("visitor need idr frame!");
            break;
        }
```

```
        default:
            break;
    }

    return rc;
}
```

5. 对讲模块退出

```
int av_talk_exit(void)
{
    return iv_avt_exit();
}
```

云存储模块

最近更新时间：2023-07-06 10:01:12

功能介绍

本模块用于将设备端的音视频数据推送并存储在云端，回看时由观看端（如 App）从云端拉取数据。

使用流程

基本推流过程

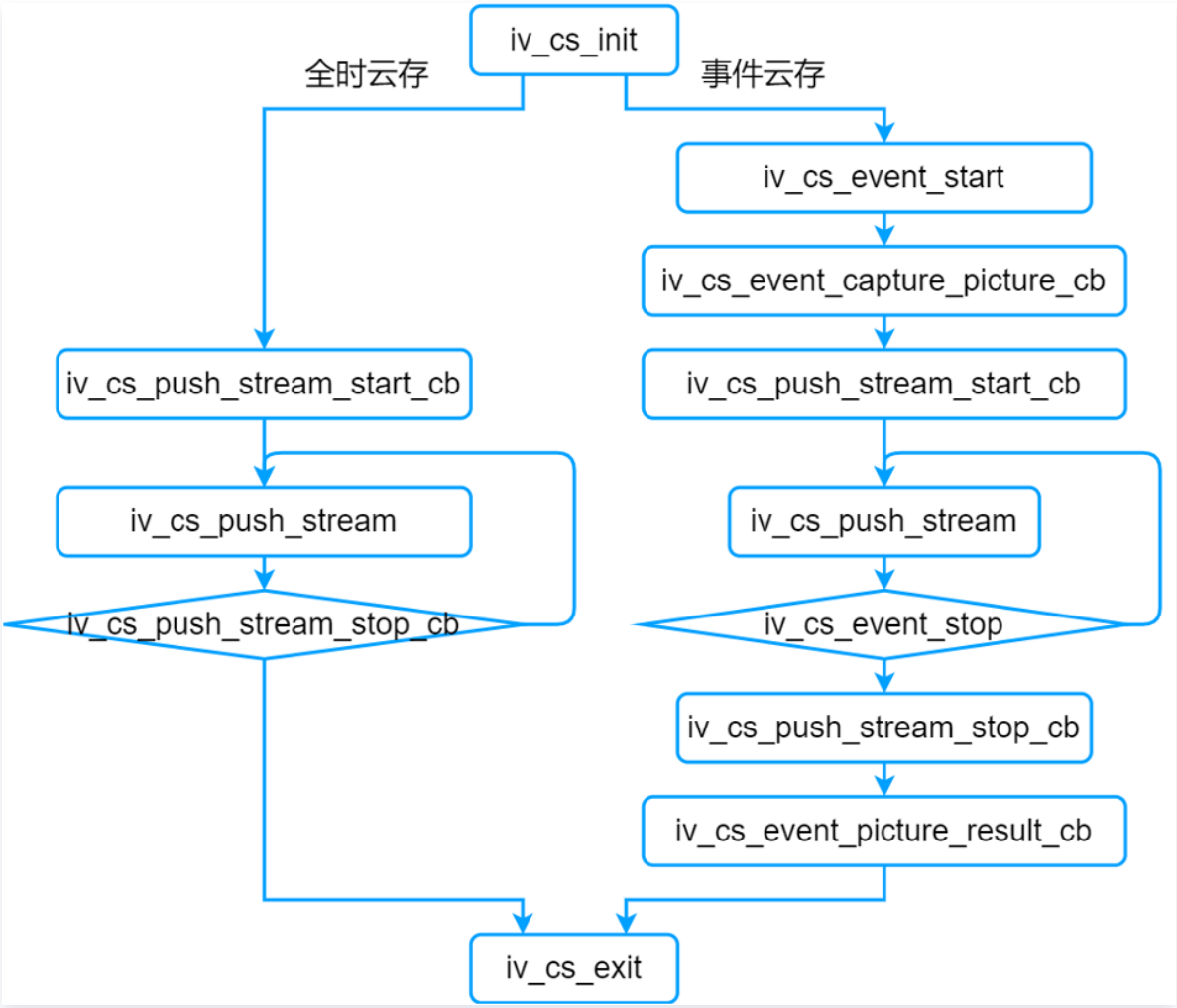
```
iv_cs_get_balance_info //获取本地套餐信息(非必须)
    |
iv_cs_init
    |
iv_cs_push_stream_start_cb // 收到回调，开始推流
    |
while(1) { iv_cs_push_stream }
    |
iv_cs_push_stream_stop_cb // 收到回调，停止推流
    |
iv_cs_exit
```

事件推流过程

```
iv_cs_get_balance_info //获取本地套餐信息(非必须)
    |
iv_cs_init
    |
iv_cs_event_start/iv_cs_event_start_ext // 用户触发事件
    |
iv_cs_event_capture_picture_cb // SDK向用户请求获取事件截图
    |
iv_cs_push_stream_start_cb // 收到回调，开始推流
    |
while(1) { iv_cs_push_stream }
    |
iv_cs_event_stop/iv_cs_event_stop_ext  //用户停止事件
    |
iv_cs_push_stream_stop_cb // 收到回调，停止推流
    |
```

```
iv_cs_event_picture_result_cb // 截图使用完毕，用户回收截图资源
    |
iv_cs_exit
```

基本流程



接口列表

该功能模块提供以下接口：

- iv_cs_init()：云存功能初始化。
- iv_cs_exit()：云存功能关闭。
- iv_cs_push_stream()：推送音、视频流。
- int iv_cs_event_start()/iv_cs_event_start_ext()：触发事件。
- int iv_cs_event_stop()/iv_cs_event_stop_ext()：停止事件。
- int iv_cs_event_directly_report()：触发即时事件上报。
- int iv_cs_get_balance_info()：获取云存套餐信息。
- int iv_cs_set_trans_time()：设置云存的传输超时时间。

- int iv_cs_set_ai_process()：设置云 AI 视频分析功能。
- int iv_cs_get_ai_process()：获取云 AI 视频分析功能。

用户需注册以下回调函数：

- (*iv_cs_push_stream_start_cb)()：音、视频流开始推流通知。
- (*iv_cs_push_stream_stop_cb)()：音、视频流停止推流通知。
- (*iv_cs_event_capture_picture_cb)()：获取事件截图。
- (*iv_cs_event_picture_result_cb)()：事件截图使用完毕通知。
- (*iv_cs_event_report_result_cb)()：事件上传状态通知。
- (*iv_cs_notify_cb)()：云存状态通知。

接口描述

iv_cs_init

功能描述

云存储模块初始化，注册相关回调，申请资源，需要模块时初始化调用。

函数原型

```
int iv_cs_init(iv_cs_init_parm_s *pstInitParm);
```

参数说明

参数名称	类型	描述	输入/输出
pstInitParm	iv_cs_init_parm_s *	云存初始化参数结构体。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_exit

功能描述

云存储模块去初始化，释放资源。

函数原型

```
int iv_cs_exit(void);
```

参数说明

参数名称	类型	描述	输入/输出
无	无	无	无

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_push_stream

功能描述

推送音视频流。

函数原型

```
int iv_cs_push_stream(iv_cm_stream_type_e eCsStreamType, void
*pstCsPack);
```

参数说明

参数名称	类型	描述	输入/输出
eCsStreamType	iv_cm_stream_type_e	云存推流类型（音频/视频）。	输入
pstCsPack	void *	云存推流的数据（音频 iv_cm_aenc_pack_s，视频 iv_cm_venc_pack_s）。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

使用说明

此接口禁止并行调用。

音频推荐使用 ADTS 的 AAC 裸流。

如使用 G711u、G711a、PCM 格式请在初始化时设置采样率、声道数等参数，对于这三种格式参数中 `iv_cs_init_parm_s.av_fmt.u32SampleNumPerFrame` 目前只支持1024采样点（双声道为2048采样点），双声道则左右声道交替存放(`LRLRLRLR...`)，以 PCM 为例：

```
int16_t audio_data[1024]; // 单声道
int16_t audio_data[2048]; // 双声道
```

视频支持 H.264 或 H.265 含 NALU 的 I帧、P帧裸流，不支持 B帧，其中I帧需包含 SPS、PPS 等数据，部分裸数据格式如下：

```
H264 I帧
00 00 00 01 67 (...) 00 00 00 01 68 (...) 00 00 00 01 65 (...)

H264 P帧
00 00 00 01 41 (...)

H265 I帧
00 00 00 01 40 (...) 00 00 00 01 42 (...) 00 00 00 01 44 (...) 00 00 00
01 2A (...)

H265 P帧
00 00 00 01 02 (...)
```

iv_cs_event_directly_report

功能描述

- 用于触发一个事件并立即上报，事件编号的合法取值为1-16，事件编号的具体含义由用户定义。
- 该接口可传入事件对应图片信息上传至云服务器，如无图片信息 pic_addr 填 NULL，pic_len 填0。
- 函数执行成功事件及对应图片可上传成功，否则上报失败。
- event_time_s 填0时 SDK 则使用系统时间。
- 未开通云存套餐的情况下可使用此接口上报只有图片的事件。

函数原型

```
int iv_cs_event_directly_report(int32_t event_id, uint32_t event_time_s,
uint8_t *pic_addr,
                                uint32_t pic_len)
```

参数说明

参数名称	类型	描述	输入/输出

event_id	int32_t	事件编号。	输入
event_time_s	uint32_t	事件触发 utc 时间，单位：秒。	输入
pic_addr	uint8_t *	事件对应图片地址。	输入
event_id	int32_t	事件对应图片长度。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_event_start

功能描述

- 用于触发一个事件，事件编号的合法取值为1-16，事件编号的具体含义由用户定义。
- 用户触发事件以后，SDK 调用 iv_cs_event_capture_picture_cb 向用户请求截图，同时会调用 iv_cs_push_stream_start_cb 通知用户开始推送音视频流。
- 截图上传完毕后 SDK 调用 iv_cs_event_picture_result_cb 通知用户图片上传结果。
- 需在上一事件停止后调用，且和 iv_cs_event_stop 成对调用。
- 适用于事件发生在调用该函数这一刻的情况。

函数原型

```
int iv_cs_event_start(int32_t event_id);
```

参数说明

参数名称	类型	描述	输入/输出
user_event_id	int32_t	事件编号	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_event_stop

功能描述

- 用于停止一个事件，事件编号的合法取值为1-16，事件编号的具体含义由用户定义。
- 用户停止事件以后，SDK 调用 iv_cs_push_stream_stop_cb 通知用户停止推送音视频流。
- 必须调用 iv_cs_event_start 后才可调，且和 iv_cs_event_start 成对调用。

函数原型

```
int iv_cs_event_stop(int32_t event_id);
```

参数说明

参数名称	类型	描述	输入/输出
user_event_id	int32_t	事件编号	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_event_start_ext

功能描述

- 用于触发一个事件，事件编号的合法取值为1-16，事件编号的具体含义由用户定义。
- 用户触发事件以后，SDK 调用 iv_cs_event_capture_picture_cb 向用户请求截图，同时会调用 iv_cs_push_stream_start_cb 通知用户开始推送音视频流。
- 截图上传完毕后SDK调用 iv_cs_event_picture_result_cb 通知用户图片上传结果。
- 适用于预录视频或补传视频的场景。
- 当 event_start_time_s 和 stream_start_time_s 小于 300 时，该值作为相对时间使用但不得小于0（小于零代表事件开始在未来，无意义），例如 event_start_time_s = 15 代表事件发生的时间是调用此函数前的15秒；大于 300 时作为绝对时间使用，即 UTC 时间。
- event_start_time_s 和 stream_start_time_s 必须同为相对时间或同为绝对时间，不得混用。
- event_start_time_s 和 stream_start_time_s 可以相同或不相同，可根据实际使用场景填写，例如 stream_start_time_s 填写 15 表示视频开始于调用此函数前的15秒，event_start_time_s 填写 0 表示事件发生于调用此函数的这一刻，这样就实现了预录视频还原完整事件经过的效果。
- event_start_time_s 和 stream_start_time_s 作为绝对时间使用时，用户需尽量保证时间准确，以免云存录像时间出现太大偏差；使用过程中尽量避免向前校时，如果校时后的时间早于最后一段视频的结束时间可能

导致已经上传的录像被覆盖。

函数原型

```
int iv_cs_event_start_ext(int32_t event_id, uint32_t event_start_time_s,
uint32_t stream_start_time_s);
```

参数说明

参数名称	类型	描述	输入/输出
user_event_id	int32_t	事件编号。	输入
event_start_time_s	uint32_t	事件开始的 utc 时间。	输入
stream_start_time_s	uint32_t	事件对应视频开始的 utc 时间。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_event_stop_ext

功能描述

- 用于停止一个事件，事件编号的合法取值为1-16，事件编号的具体含义由用户定义。
- 用户停止事件以后，SDK 调用 iv_cs_push_stream_stop_cb 通知用户停止推送音视频流。
- 必须调用 iv_cs_event_start_ext 后才可调，且和iv_cs_event_start_ext 成对调用。
- 当 event_stop_time_s 或 stream_stop_time_s 小于 300 时，该值作为相对时间使用但不得小于0（小于零代表事件结束在未来，无意义），例如 event_stop_time_s = 15 代表事件结束的时间是调用此函数前的15秒；大于 300 时作为绝对时间使用，即 UTC 时间。
- event_stop_time_s 和 stream_stop_time_s 必须同为相对时间或同为绝对时间，不得混用。
- event_stop_time_s 和 stream_stop_time_s 作为绝对时间使用时，用户需尽量保证时间准确，以免云存录像时间出现太大偏差；使用过程中尽量避免向前校时，如果校时后的时间早于最后一段视频的结束时间可能导致已经上传的录像被覆盖。

函数原型

```
int iv_cs_event_stop_ext(int32_t event_id, uint32_t event_stop_time_s,
uint32_t stream_stop_time_s);
```

参数说明

参数名称	类型	描述	输入/输出
user_event_id	int32_t	事件编号。	输入
event_stop_time_s	uint32_t	事件结束的 utc 时间。	输入
stream_stop_time_s	uint32_t	事件对应视频结束的 utc 时间。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_get_balance_info

功能描述

- 用于获取当前设备的云存的套餐信息。
- 当 `timeout_s` 大于0时，该接口为阻塞获取信息，直接从后台获取最新套餐信息，失败返回错误码 `IV_ERR_CS_QUERY_SERVICE_TIMEOUT` ，成功返回 `IV_ERR_NONE` 和套餐信息。
- 当 `timeout_s` 等于0时，该接口为非阻塞接口，直接返回 `SDK` 内的套餐缓存信息。
- 当云存的套餐在后台发生变化后，正常情况下会及时更新到设备并缓存在 SDK 内，通过本接口直接查询 SDK 内缓存套餐信息效率更高。
- 接口可在设备上线后云存 `SDK` 未初始化前进行调用来获取套餐信息，云存 SDK 未初始化时采用轮询方式调用该接口，建议只在第一次调用时采用阻塞方式，后续的调用采用非阻塞方式，否则会引起带宽资源浪费。

函数原型

```
int iv_cs_get_balance_info(iv_cs_balance_info_s *pstBalanceInfo,
uint32_t timeout_s)
```

参数说明

参数名称	类型	描述	输入/输出
pstBalanceInfo	iv_cs_balance_info_s	云存套餐信息。	输出
timeout_s	uint32_t	接口获取超时时间(单位：秒)。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_set_ai_process

功能描述

- 设置需要进行 AI 分析的事件 ID 以及 AI 分析的类型。
- 可以在云存初始化之后、云存退出前任意时刻调用，如需长期使能 AI，建议在云存初始化后进行设置，在云存退出前关闭，减少在事件触发时频繁开关。

函数原型

```
int iv_cs_set_ai_process(int32_t event_id, iv_cs_ai_type_e ai_type, void
*args);
```

参数说明

参数名称	类型	描述	输入/输出
event_id	int32_t	需要进行 AI 分析的事件 ID。	输入
ai_type	iv_cs_ai_type_e	需要进行 AI 分析的类型。	输入
arg	void *	用户参数，暂未使用，填 NULL。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_get_ai_process

功能描述

获取某个事件的 AI 分析类型。

函数原型

```
int iv_cs_get_ai_process(int32_t event_id, iv_cs_ai_type_e *ai_type,
void *args);
```

参数说明

参数名称	类型	描述	输入/输出
event_id	int32_t	需要读取的事件 ID。	输入
ai_type	iv_cs_ai_type_e *	事件的 AI 分析类型。	输入
arg	void *	用户参数，暂未使用，填 NULL。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_set_trans_time

功能描述

分别设置云存传输的上传和接收最大超时时间，单位毫秒，不能设置为0。

函数原型

```
int iv_cs_set_trans_time(uint32_t upload_timeout_ms, uint32_t
reply_timeout_ms)
```

参数说明

参数名称	类型	描述	输入/输出
upload_timeout_ms	uint32_t	上传最大超时时间，单位毫秒。	输入
reply_timeout_ms	uint32_t	接口最大超时时间，单位毫秒。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_push_stream_start_cb

功能描述

开始推流回调，事件触发后、云存套餐开通后等情况会触发此回调，收到此回调后方可开始推流。

函数原型

```
int (*iv_cs_push_stream_stop_cb)(void);
```

参数说明

无

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_push_stream_stop_cb

功能描述

停止推流回调，事件结束、云存服务到期等情况会触发此回调，收到此回调后停止推流。

函数原型

```
int (*iv_cs_push_stream_start_cb)(void);
```

参数说明

无

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_event_capture_picture_cb

功能描述

抓图回调，建议非阻塞使用，仅支持 jpg 格式，返回时提供图片指针和大小，如果不需抓图请填 NULL 和0。

函数原型

```
int (*iv_cs_event_capture_picture_cb)(int32_t event_id, uint8_t **pic,
int32_t *size);
```

参数说明

参数名称	类型	描述	输入/输出
event_id	int32_t	用户触发的事件 id，根据不同 id 可以上传对应事件的图片。	输入
pic	uint8_t **	指向保存截图的内存。	输出
size	int32_t *	指向截图的大小。	输出

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_event_picture_result_cb

功能描述

上传图片结束回调，并通知上传结果。

每张抓图都会回调一次，在收到对应图片回调前不得回收图片资源，以免 SDK 上传图片失败。

函数原型

```
int (*iv_cs_event_picture_result_cb)(uint8_t **pic, iv_err_code_e
err_code);
```

参数说明

参数名称	类型	描述	输入/输出
pic	uint8_t **	指向保存截图的内存，与 iv_cs_event_capture_picture_cb 的入参对应。	输入
err_code	iv_err_code_e	上传图片的结果。	输入

返回值

返回值	描述

IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_event_report_result_cb

功能描述

用户事件上报结果回调，用户结束事件后事件上报的结果通过该回调通知。

函数原型

```
int (*iv_cs_event_report_result_cb)(iv_cs_event_result_info
*pst_result_info);
```

参数说明

参数名称	类型	描述	输入/输出
pst_result_info	iv_cs_event_result_info *	指向事件上报结果的信息。	输出

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_cs_notify_cb

功能描述

云存消息通知回调，目前包括云存视频分段的上传情况通知、内部上传连接情况通知、内部发送缓存水位告警通知消息等。

- notify_msg_type 为 IV_CS_AV_UPLOAD_STATE_MSG 时，pst_notify_data 使用联合体成员 iv_cs_upload_info_s。
- notify_msg_type 为其余消息时，pst_notify_data 为 NULL。

函数原型

```
int (*iv_cs_notify_cb)(iv_cs_notify_msg_type_e notify_msg_type,
                       iv_cs_notify_msg_data *pst_notify_data);
```

参数说明

参数名称	类型	描述	输入/输出
notify_msg_type	iv_cs_notify_msg_type_e	云存通知消息类型。	输入
pst_notify_data	iv_cs_notify_msg_data*	云存通知消息数据。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

数据结构列表

云存模块涉及以下数据结构：

- iv_cm_*：请参考音视频结构公共模块。
- iv_cs_type_e：云存套餐类型枚举。
- iv_cs_balance_info_s：云存套餐状态结构体。
- iv_cs_ai_type_e：云 AI 分析的类型枚举。
- iv_cs_event_report_opt_e：事件云存上传类型枚举。
- iv_cs_upload_state_e：云存上传状态枚举。
- iv_cs_upload_slice_info_s：云存某段视频上传状态结构体。
- iv_cs_upload_info_s：云存视频上传状态结构体。
- iv_cs_notify_msg_type_e：云存回调消息类型。
- iv_cs_notify_msg_data：云存回调消息联合体。
- iv_cs_event_result_info：云存事件上传结果结构体。
- iv_cs_congestion_ctrl_s：云存水位控制结构体。
- iv_cs_init_parm_s：云存模块初始化参数结构体。

数据结构描述

iv_cs_type_e

功能描述

云存套餐类型。

结构原型

```
typedef enum
{
```

```
    CS_TYPE_NONE      = 0,
    CS_TYPE_FULL_TIME = 1,
    CS_TYPE_EVENT     = 2,
} iv_cs_type_e;
```

参数说明

成员名称	描述	取值
CS_TYPE_NONE	无套餐	0
CS_TYPE_FULL_TIME	全时套餐	1
CS_TYPE_EVENT	事件套餐	2

iv_cs_balance_info_s

功能描述

云存套餐结构体。

结构原型

```
typedef struct iv_cs_balance_s {
    int cs_switch;
    iv_cs_type_e cs_type;
} iv_cs_balance_info_s;
```

参数说明

成员名称	描述	取值
cs_switch	云存开关	int
cs_type	云存套餐类型	iv_cs_type_e

iv_cs_ai_type_e

功能描述

云 AI 类型。

结构原型

```
typedef enum
{
    CS_AI_TYPE_CLOSE       = (uint32_t)0,
    CS_AI_TYPE_FALL_DETECT = (uint32_t)(1 << 0),
```

```
} iv_cs_ai_type_e;
```

参数说明

成员名称	描述	取值
CS_AI_TYPE_CLOSE	关闭云 AI。	(uint32_t)0
CS_AI_TYPE_FALL_DETECT	老人跌倒检测云 AI。	(uint32_t)(1 << 0)

iv_cs_event_report_opt_e

功能描述

事件云存上传类型枚举，用于告知 SDK 所有事件按视频事件处理或按图片事件处理，默认为0。

- 设置为 CS_EVENT_TS_REQ 时，至少成功上传一段视频 SDK 才认为这是一个完整事件。
- 设置为 CS_EVENT_PIC_REQ 时，成功上传截图后 SDK 才认为这是一个完整事件。
- 设置为 CS_EVENT_TS_OR_PIC_REQ 时，成功上传一段视频或成功上传截图 SDK 都认为这是一个完整事件。

> 例如：假设用户要求所有事件必须有视频，建议设置为 CS_EVENT_TS_REQ；假设用户只需上传事件截图不需上传视频，建议设置为 CS_EVENT_PIC_REQ。

结构原型

```
typedef enum cs_event_upload_options
{
    CS_EVENT_TS_REQ = 0,
    CS_EVENT_PIC_REQ = 1,
    CS_EVENT_TS_OR_PIC_REQ = 2,
} iv_cs_event_report_opt_e;
```

参数说明

成员名称	描述	取值
CS_EVENT_TS_REQ	成功上传视频，图片可选上传。	0
CS_EVENT_PIC_REQ	成功上传截图，视频可选上传。	1
CS_EVENT_TS_OR_PIC_REQ	成功上传视频或图片。	2

iv_cs_upload_state_e

功能描述

视频是否上传成功。

结构原型

```
typedef enum
{
    IV_CS_UPLOAD_OK  = 0,
    IV_CS_UPLOAD_ERR = 1,
} iv_cs_upload_state_e;
```

参数说明

成员名称	描述	取值
IV_CS_UPLOAD_OK	上传成功	0
IV_CS_UPLOAD_ERR	上传失败	1

iv_cs_upload_slice_info_s

功能描述

通知用户某段视频的上传状态。

用户可根据其中的 UTC 时间、PTS、帧序号等信息得知视频上传的状态。此结构体内记录的信息均为视频帧。一般而言只有 upload_size == total_size 且 state == IV_CS_UPLOAD_OK 才认为视频上传完全正常。

结构原型

```
typedef struct {
    uint64_t utc_sec_a;
    uint64_t utc_sec_b;
    uint64_t pts_ms_a;
    uint64_t pts_ms_b;
    uint32_t frame_seq_a;
    uint32_t frame_seq_b;
    uint32_t upload_size;
    uint32_t total_size;
    iv_cs_upload_state_e state;
} iv_cs_upload_slice_info_s;
```

参数说明

成员名称	描述	取值
utc_sec_a	本段视频开始的 UTC 时间（SDK 内部生成）。	uint64_t

utc_sec_b	本段视频结束的 UTC 时间（SDK 内部生成）。	uint64_t
pts_ms_a	本段视频第一帧的 PTS（来自 iv_cs_push_stream）。	uint64_t
pts_ms_b	本段视频最后一帧的 PTS（来自 iv_cs_push_stream）。	uint64_t
frame_seq_a	本段视频第一帧的序号（来自 iv_cs_push_stream）。	uint32_t
frame_seq_b	本段视频最后一帧的序号（来自 iv_cs_push_stream）。	uint32_t
upload_size	本段视频上传成功的大小（估算值）。	uint32_t
total_size	本段视频总大小（精确值）。	uint32_t
state	本段视频的上传状态（成功/失败）。	iv_cs_upload_state_e

iv_cs_upload_info_s

功能描述

通知用户视频的上传状态，由若干 iv_cs_upload_slice_info_s 组成。

结构原型

```
typedef struct {
    iv_cs_upload_slice_info_s slice_info[MAX_UPLOAD_INFO_NUM];
    int32_t num;
} iv_cs_upload_info_s;
```

参数说明

成员名称	描述	取值
slice_info	某段视频的上传状态。	iv_cs_upload_slice_info_s
num	slice_info 的数量。	int32_t

iv_cs_notify_msg_type_e

功能描述

云存回调消息类型。

结构原型

```
typedef enum cs_notify_type
{
    IV_CS_AV_UPLOAD_CONNECT_FAIL = 0,
    IV_CS_AV_UPLOAD_CONNECT_RECOVER = 1,
    IV_CS_AV_UPLOAD_STATE_MSG = 2,
    IV_CS_AV_UPLOAD_WATERMARK_LOW = 3,
    IV_CS_AV_UPLOAD_WATERMARK_WARN = 4,
    IV_CS_AV_UPLOAD_WATERMARK_HIGH = 5,
    IV_CS_COS_SERVICE_WRONG = 6,
    IV_CS_MSG_BUTT
} iv_cs_notify_msg_type_e;
```

参数说明

成员名称	描述	取值
IV_CS_AV_UPLOAD_CONNECT_FAIL	云存服务器连接失败。	0
IV_CS_AV_UPLOAD_CONNECT_RECOVER	云存服务器连接恢复。	1
IV_CS_AV_UPLOAD_STATE_MSG	云存上传状态消息。	2
IV_CS_AV_UPLOAD_WATERMARK_LOW	云存低水位告警。	3
IV_CS_AV_UPLOAD_WATERMARK_WARN	云存中水位告警。	4
IV_CS_AV_UPLOAD_WATERMARK_HIGH	云存高水位告警。	5
IV_CS_COS_SERVICE_WRONG	云存服务不可用。	6
IV_CS_MSG_BUTT	-	-

iv_cs_notify_msg_data

功能描述

iv_cs_notify_cb 回调携带的消息联合体。

结构原型

```
typedef union {
    iv_cs_upload_info_s *av_result_info;
} iv_cs_notify_msg_data;
```

参数说明

成员名称	描述	取值
av_result_info	视频上传的结果。	iv_cs_upload_info_s

iv_cs_event_result_info

功能描述

事件的上传结果。

结构原型

```
typedef struct {
    int32_t result_code;
    int32_t event_id;
    uint32_t start_time_s;
    uint32_t end_time_s;
} iv_cs_event_result_info;
```

参数说明

成员名称	描述	取值
result_code	上报结果。	0：失败，1：成功。
event_id	事件序号。	1到16。
start_time_s	事件开始时间，单位秒。	UTC 时间戳。
end_time_s	事件结束时间，单位秒。	UTC 时间戳。

iv_cs_congestion_ctrl_s

功能描述

设置云存缓存水位告警相关参数。

结构原型

```
typedef struct {
    size_t low_mark;
    size_t warn_mark;
    size_t high_mark;
    int8_t enable;
} iv_cs_congestion_ctrl_s;
```

参数说明

成员名称	描述	取值
low_mark	设置触发告警的低水位值，单位字节。	不大于 u32MaxGopSize
warn_mark	设置触发告警的低水位值，单位字节。	不大于 u32MaxGopSize
high_mark	设置触发告警的低水位值，单位字节。	不大于 u32MaxGopSize
enable	设置是否开启水位值告警功能。	0：否，1：是

iv_cs_init_parm_s

功能描述

云存模块初始化参数结构体。

结构原型

```
typedef struct iv_cs_init_parm_s
{
    uint32_t u32MaxGopSize;
    iv_cs_congestion_ctrl_s congestion_cfg;
    iv_cm_av_data_info_s av_fmt;
    iv_cs_event_report_opt_e event_report_opt;
    int (*iv_cs_push_stream_start_cb)(void);
    int (*iv_cs_push_stream_stop_cb)(void);
    void (*iv_cs_ai_service_notify_cb)(unsigned int ai_server_type,
unsigned long long utc_expire);
    int (*iv_cs_event_capture_picture_cb)(int32_t event_id, uint8_t
**pic, int32_t *size);
    int (*iv_cs_event_picture_result_cb)(uint8_t **pic, int32_t
err_code);
    int (*iv_cs_event_report_result_cb)(iv_cs_event_result_info
*pst_result_info);
    int (*iv_cs_notify_cb)(iv_cs_notify_msg_type_e notify_msg_type,
                           iv_cs_notify_msg_data *pst_notify_data);
}iv_cs_init_parm_s;
```

参数说明

成员名称	描述	取值
u32MaxGopSize	云存缓存大小，单位字节，取值为4KB的整数倍。	uint32_t
congestion_cfg	视频上传拥塞控制信息。	iv_cs_congestion_ctrl_s

av_fmt	云存音视频格式（如果云存视频采用动态帧率，其中的u32Framerate 请填0）。	iv_cm_av_data_info_s
event_report_opt	事件上报选项设置。	iv_cs_event_report_opt
iv_cs_push_stream_start_cb	开始推流回调（详细说明见上文）。	无
iv_cs_push_stream_stop_cb	停止推流回调（详细说明见上文）。	无
iv_cs_ai_service_notify_cb	云AI服务状态回调（暂未使用）。	NULL
iv_cs_event_capture_picture_cb	获取事件截图回调（详细说明见上文）。	无
iv_cs_event_picture_result_cb	图片上传结果回调（详细说明见上文）。	无
iv_cs_event_report_result_cb	事件上报结果回调（详细说明见上文）。	无
iv_cs_notify_cb	云存模块消息通知回调（详细说明见上文）。	无

注意事项

1. 云存开始推流送入的第一帧尽量保证为 IDR 帧，否则回放时可能导致花屏、解码失败等问题。
2. 云存可能因网络波动上传失败，使用过程请尽量保持网络状况良好。
3. u32MaxGopSize 取值推荐能保存一个 GOP 的视频和对应的音频，假设帧率为20fps，GOP 为80帧，经统计4秒的视频数据约700KB，音频约60KB，则 u32MaxGopSize 可以取 800KB。
4. u32MaxGopSize 如果设备内存紧张可以适当减小，如减小至原来的80%（上述为例即640KB），但不得小于一个I帧的大小，取值过小会导致云存上传更难以承受网络波动。
5. 用户可以在未收到 iv_cs_push_stream_stop_cb 回调的情况下主动停止推送云存录像，SDK 超过120秒未收到音视频数据会暂停云存录像。
6. 云存使用过程中禁止修改音视频格式，如格式发生变化，请调用 iv_cs_exit 关闭云存，再调用 iv_cs_init 重新初始化云存并设置新的音视频格式。
7. 云存功能内存开销计算：
   - 输入 aac 格式音频：内存开销 = u32MaxGopSize + 100KB（SDK 内部数据）。
   - 输入其他格式音频：内存开销 = u32MaxGopSize + 120KB（单声道）或 160KB（双声道）+ 100KB（SDK 内部数据）。

8. 事件云存触发建议事件至少持续3s及以上以保证视频片段、图片可及时上传，不会发生丢弃；否则可能会发生视频、图片上传失败，只上报事件触发消息等类似情况。
9. 单个事件持续时间建议不要超过1小时，尽量控制在10分钟以内，如果持续时间确实很长建议进行分割。
10. 事件云存支持并行事件，同一时刻允许触发多个不同事件，但不能触发多个相同事件。
11. 事件云存支持预录视频，网络良好的情况允许预录视频快速上传来追赶实时视频。
12. 并行事件只有第一个触发的事件支持预录视频，后续并行事件调用 iv_cs_event_start_ext()，stream_start_time_s 请填0或当前 UTC 时间；或直接调用 iv_cs_event_start()。
13. 如果没有套餐，全时云存不会触发推流通知, 此时推流也会返回错误码 IV_ERR_CS_APPLY_NO_SERVICE，需要用户定时检查套餐情况。
    - 当全时云存套餐生效时，会触发开始推流通知，当无效时，会触发停止推流通知。
    - 当事件云存套餐生效时，不会有通知，但此时调用事件云存的相关接口可以正常工作。
14. 事件支持无图片事件，当事件无图片时 SDK 不会调用 iv_cs_event_picture_result_cb 释放图片资源。
15. 事件云存强烈建议不要上传未来的视频，SDK 可以接受小幅度的时钟误差（建议正负1分钟以内）。系统时钟快于现实时间这种情况在使用中要特别注意，以免导致视频被覆盖。假设这样一个场景：现实时间为 09:00:00，设备端的系统时钟为09:00:40，此时触发了一个10秒的事件，并上传视频，即视频的时间为 09:00:40-09:00:50，之后设备开始校时，将时间修改为 09:00:10，在 09:00:30 触发了一个30秒的事件，并上传视频，即视频的时间为 09:00:30-09:01:00，此时第二个事件的视频会覆盖第一个事件的视频。
16. 云存消息通知回调 iv_cs_notify_cb 需要支持重入。

示例代码

```
// 云存推流
while (run_flag)
{
    // 调用者主动推流，由调用者保证推送音视频帧的顺序与间隔，否则可能造成音视频不同步
    if(compare_time_stamp(frame_count_audio, audio_time_base,
frame_count_video, video_time_base) <= 0)
    {
        frame_count_audio += 1;
        get_aac_frame(audio_frame_buf, &audio_len);
        audio_pack.pu8Addr = audio_frame_buf;
        audio_pack.u32Len = audio_len;
        audio_pack.u64TimeStamp = (uint64_t)(frame_count_a * 1024 /
48.0);
        audio_pack.u32Seq = frame_count_a;
        iv_cs_push_stream(IV_CM_STREAM_TYPE_AUDIO, &audio_pack);
    }
    else
    {
        frame_count_video += 1;
```

```
            get_video_frame_h264(video_frame_buf, &video_len, &frame_type);
            video_pack.pu8Addr = video_frame_buf;
            video_pack.u32Len = video_len;
            video_pack.eFrameType = frame_type;
            video_pack.u64PTS = (uint64_t)(frame_count_v * 1000 / 25.0);
            video_pack.u32Seq = frame_count_v;
            iv_cs_push_stream(IV_CM_STREAM_TYPE_VIDEO, &video_pack);
        }
        sleep_ms(10);
    }
```

详细例程请参考 `samples\samples\cloud_storage\cloud_storage.c` **及** `samples\samples\` **目录内的其他代码。**

自定义信令模块

最近更新时间：2024-08-23 17:43:51

本模块提供自定义信令与后台服务器数据收发功能。

功能介绍

本模块在系统模块、物模型模块初始化完成后进行初始化，提供服务器下行数据接收回调入口，提供自定义数据向后台服务器发送入口，从而以实现设备侧自定义信令与服务器的交互。

使用流程

模块初始化 > 发送数据 / 接收数据 > 模块去初始化。

接口参考

该功能模块提供以下接口：

- iv_uc_init：自定义模块初始化。
- iv_uc_exit：自定义模块去初始化。
- iv_uc_send_msg：自定义信令发送。

用户需注册以下回调函数

iv_uc_recv_msg_cb：自定义信令接收回调。

iv_uc_init

功能描述

自定义信令模块的初始化。包括自定义信令接口回调注册、消息通信初始化等。

函数原型

```
int iv_uc_init(iv_uc_init_parm_s *pstInitParm);
```

参数说明

参数名称	类型	描述	输入/输出
pstInitParm	iv_uc_init_parm_s	初始化参数	输入

返回值

返回值	描述

IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_uc_exit

功能描述

自定义信令模块去初始化。

函数原型

```
int iv_uc_exit(void);
```

参数说明

无

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

iv_uc_send_msg

功能描述

发送自定义信令。

函数原型

```
uint32_t iv_uc_send_msg(void *pUcData, uint32_t u32UcDataLen);
```

参数说明

参数名称	类型	描述	输入/输出
pUcData	void *	自定义信令数据。	无
u32UcDataLen	uint32_t	自定义信令数据长度。	无

返回值

返回值	描述
IV_ERR_NONE	成功。

IV_ERR_*	失败，对应相应错误码。

iv_uc_recv_msg_cb

功能描述

自定义信令接收回调。

函数原型

```
void (*iv_uc_recv_msg_cb)(void *pUcData, uint32_t u32UcDataLen);
```

参数说明

参数名称	类型	描述	输入/输出
pUcData	void *	自定义信令数据地址。	输入
u32UcDataLen	uint32_t	自定义信令数据长度。	输入

返回值

无

数据结构

该模块提供以下数据结构：

iv_uc_init_parm_s：初始化参数。

iv_uc_init_parm_s

功能描述

初始化参数。

结构原型

```
typedef struct iv_uc_init_parm_s
{
    void (*iv_uc_recv_msg_cb)(void *pUcData, uint32_t u32UcDataLen);
}iv_uc_init_parm_s;;
```

参数说明

成员名称	描述	取值
iv_uc_recv_msg_cb	自定义信令数据接收回调。	–

注意事项

1. 该模块初始化需要在系统模块、物模型模块初始化完成后才能进行模块初始化，初始化成功后即可调用发送数据接口或接收数据回调进行自定义信令的收发。
2. 自定义信令发送和接收的实际消息长度以接口数据长度入参为准。

示例代码

1. 自定义信令模块初始化

```
int user_cmd_init(void)
{
    int eErrCode = 0;

    iv_uc_init_parm_s stUCInitParm;
    stUCInitParm.iv_uc_recv_msg_cb        = userCmd_recv_msg;
    eErrCode                              = iv_uc_init(&stUCInitParm);
    if (eErrCode < 0) {
        Log_e("iv_uc_init error:%d", eErrCode);
    }
    return eErrCode;
}
```

2. 自定义信令模块退出

```
int iv_uc_exit(void)
```

3. 自定义信令接收

```
static void userCmd_recv_msg(void *pUcData, uint32_t u32UcDataLen)
{
    char msgbuf[1024] = {0};

    if (u32UcDataLen >= sizeof(msgbuf)) {
        Log_e("recv msg buf is not enough, datalen:%u", u32UcDataLen);
        return;
    }

    memcpy(msgbuf, pUcData, u32UcDataLen);

    return;
```

```
}
```

AI 模块

最近更新时间：2024-08-23 17:43:51

功能介绍

本模块主要实现用户将需要检测的图片上传至 COS，并请求 AI 后台完成推理，实现用户数据上云 AI 检测功能。

使用流程

```
iv_ai_init
    |
iv_ai_notify_event
    |
iv_ai_start/iv_ai_stop
    |
iv_ai_add_model_id / iv_ai_del_mode_id
    |
iv_ai_upload_cos_result_cb // 收到sdk 处理回调
    |
iv_ai_deinit
```

接口参考

该功能模块提供以下接口：

- iv_ai_init()：AI 模块初始化。
- iv_ai_deinit()：AI 系统去初始化。
- iv_ai_start()：启动 AI 推理。
- iv_ai_stop()：停止 AI 推理。
- iv_ai_set_notify_event()：通知 SDK 发生抓图等事件。
- iv_ai_add_model_id()：添加模型 ID。
- iv_ai_del_model_id()：删除模型 ID。

> ⚠ **用户需注册以下回调函数**
> (*iv_ai_upload_cos_result_cb)()：SDK 结果处理回调。

iv_ai_init

函数原型

```
void *iv_ai_init(iv_ai_init_parm_s params);
```

功能描述

进行 AI 功能初始化的接口函数，返回 AI 处理的 handle。

参数说明

参数名称	类型	描述	输入/输出
params	iv_ai_init_parm_s	初始化 AI 需要传入的参数。	输入

返回值

初始化成功，则返回对应的 handle，失败返回 NULL。

iv_ai_deinit

函数原型

```
int iv_ai_deinit(void **pp_handle);
```

功能描述

销毁 AI 功能。

参数说明

参数名称	类型	描述	输入/输出
pp_handle	void **	通过 iv_ai_init 初始化的 handle。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应 错误码 。

iv_ai_start

函数原型

```
int iv_ai_start(void *handle);
```

功能描述

开始 AI 服务。

参数说明

参数名称	类型	描述	输入/输出
handle	void *	通过 iv_ai_init 初始化的 handle。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应 错误码 。

iv_ai_stop

函数原型

```
int iv_ai_stop(void *handle);
```

功能描述

停止 AI 服务。

参数说明

参数名称	类型	描述	输入/输出
handle	void *	通过 iv_ai_init 初始化的 handle。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应 错误码 。

iv_ai_notify_event

函数原型

```
int iv_ai_notify_event(void *handle, int evt_id, void *arg);
```

功能描述

通知 SDK 发生了相关的事件，例如抓图事件。

参数说明

参数名称	类型	描述	输入/输出
handle	void *	通过 iv_ai_init 初始化的 handle。	输入
evt_id	int	事件 ID，目前只支持抓图事件（IV_AI_EVENT_PIC_TRIG）。	输入
arg	void *	事件相关的参数，例如抓图事件需要告知抓图的 ID，以便 SDK 获取图片路径。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应 错误码 。

iv_ai_add_model_id

函数原型

```
int iv_ai_add_model_id(void *handle, int model_id);
```

功能描述

通过此接口动态添加 AI 的模型 ID。

参数说明

参数名称	类型	描述	输入/输出
handle	void *	通过 iv_ai_init 初始化的 handle。	输入
model_id	int	模型 ID，需要添加的推理模型 ID。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应 错误码 。

iv_ai_del_model_id

函数原型

```
int iv_ai_del_mode_id(void *handle, int model_id);
```

功能描述

通过此接口动态删除 AI 的模型 ID。

参数说明

参数名称	类型	描述	输入/输出
handle	void *	通过 iv_ai_init 初始化的 handle。	输入
model_id	int	模型 ID，需要添加的推理模型 ID。	输入

返回值

返回值	描述
IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应 错误码 。

iv_ai_upload_cos_result_cb

函数原型

```
int32_t (*iv_ai_upload_cos_result_cb)(char *file_path, int err_code);
```

功能描述

- 上传图片结束回调，并通知上传结果。
- 在收到对应图片回调前不得回收图片资源，否则 SDK 上传图片可能失败。

参数说明

参数名称	类型	描述	输入/输出
file_path	上传图片绝对路径	上传对应图片的文件路径。	输出
err_code	int	上传图片的结果。	输出

返回值

无

数据结构

该模块提供以下数据结构：

iv_ai_init_parm_s：初始化 AI 需要传入的参数 。

iv_ai_init_parm_s

参数说明

名称	类型	描述
work_dir	char *	需要 AI 推理图片存储路径，需要 AI 推理图片存储路径，长度限定为128字节。
model_id	int	AI 模型 ID，目前只支持1 – 128，1为人形检测模型。
iv_ai_upload_cos_result_cb	void (*) (char *, int)	用来处理 COS 上传的结果，参数为上传 COS 的图片路径和上传的结果。

返回值

初始化成功，则返回对应的 handle，失败返回 NULL。

注意事项

上传图片要求如下：

- 图片 base64 编码后大小不可超过5M。
- 图片分辨率不得超过 1920 * 1080 。
- 建议图片存储于腾讯云，从而保障 URL 更高的下载速度和稳定性；非腾讯云存储的 URL 速度和稳定性可能受一定影响。
- 支持 PNG、JPG、JPEG、BMP，不支持 GIF 图片。

示例代码

```
//AI模块初始化
```

```
#define CLOUDAI_TEST_ID      (1)
#define cloudai_work_dir      ("./demo_media/ai_pic/")
static void test_upload_cos_result_cb(char *file_path, int result)
{
    if(IV_AI_UPLOAD_COS_SUCCESS == result) {
        printf("Upload file [%s] to COS successful!\n", file_path);
    } else {
        printf("Failed to upload file [%s]!\n", file_path);
    }
}

iv_ai_init_parm_s params = {cloudai_work_dir, CLOUDAI_MODEL_ID,
test_upload_cos_result_cb};

ai_handle = iv_ai_init(params);
if (!ai_handle) {
    Log_d("Failed to initialize cloudai client");
    return IV_ERR_CLOUDAI_INIT_FAIL;;
}

//通知sdk抓图
#define CLOUDAI_TEST_ID      (1)

int cur_id = CLOUDAI_TEST_ID;
ret = iv_ai_notify_event(ai_handle, IV_AI_EVENT_PIC_TRIG, &cur_id);
if(0 != ret) {
    Log_e("send event failed!");
}
```

低功耗保活模块

最近更新时间：2023-05-23 17:12:44

低功耗设备使用低功耗保活模块使设备在主控断电或深度睡眠后自主维持保活连接，并支持远程唤醒。

功能介绍

本模块提供设备进入低功耗保活的前置工作，通过调用该模块会使设备发起和保活服务器的保活连接，建立保活链路。通过将成功建立的链路以及唤醒和维持心跳所需的数据传递给应用层，以便于应用层进行其他处理;该模块是 IoT Video 的可选模块。

流程

进入保活

```
iv_sys_init() //系统初始化
    |
iv_dm_init() //物模型初始化
    |
iv_sys_online_cb() //设备上线通知
    |
  working     //正常工作
    |
iv_get_keep_alive_info() //获取保活信息
    |
iv_dm_exit() //物模型退出
    |
iv_sys_exit() //系统退出
    |
调用硬件建立保活链路
```

唤醒

```
远端下发唤醒指令
    |
服务器发送唤醒消息
    |
硬件保活模块
    |
输出唤醒信号
    |
```

```
唤醒主控
```

保活

```
1.getaddrinfo, 根据域名查询IP地址
    |
2.connect, 根据IP和端口号建立连接
    |
3. send鉴权信息
    |
4. recv后台消息
    |
5.是否有唤醒消息——如果有，退出保活，唤醒主控；否则继续6
    |
6. send保活信息
    |
跳转到步骤4
```

接口参考

该功能模块提供以下接口：
iv_get_keep_alive_info：获取保活信息。

iv_get_keep_alive_info

接口描述

向后台请求保活信息。

```
int iv_get_keep_alive_info(iv_keep_alive_info_s* pkeep_alive_info, int
timeout_ms);
```

参数说明

参数名称	类型	描述	输入/输出
pkeep_alive_info	iv_keep_alive_info_s	保活信息。	输入
timeout_ms	int	超时时间，单位 ms。	输入

返回值

返回值	描述

IV_ERR_NONE	成功。
IV_ERR_*	失败，对应相应错误码。

数据结构

本模块提供以下数据结构：

iv_keep_alive_info_s：保活信息。

iv_keep_alive_info_s

描述

保活信息内容。

```
typedef struct {
    //服务器域名
    char server_addr[KEEP_ALIVE_MESSAGE_MAX_LEN * 2];
    //服务器端口号
    uint32_t port;
    //鉴权消息
    uint8_t auth_msg[KEEP_ALIVE_MESSAGE_MAX_LEN];
    //鉴权消息长度
    uint32_t auth_msg_len;
    //心跳消息
    uint8_t heart_beat_msg[KEEP_ALIVE_MESSAGE_MAX_LEN];
    //心跳消息长度
    uint32_t heart_beat_msg_len;
    //唤醒消息
    uint8_t wake_up_msg[KEEP_ALIVE_MESSAGE_MAX_LEN];
    //唤醒消息长度
    uint32_t wake_up_msg_len;
} iv_keep_alive_info_s;
```

参数说明

成员名称	描述
server_addr	服务器域名。
port	端口号。
auth_msg	鉴权消息。
auth_msg_len	鉴权消息长度。

heart_beat_msg	心跳信息。
heart_beat_msg_len	心跳消息长度。
wake_up_msg	唤醒消息。
wake_up_msg_len	唤醒消息长度。

使用说明

1. 设备保活状态和在线状态不能同时存在，否则会造成设备强制下线。
2. 只有设备上线时，才能获取到保活信息，只有设备离线后，才可以进入保活状态。
3. 心跳包最大发送间隔为10min，推荐1min发送一次。

示例代码

1. 获取保活信息。

```
int keep_alive_info_update(void)
{
 int timeout_ms = 5000;
 int ret        = 0;

 memset(&sg_keep_alive_cfg, 0, sizeof(iv_keep_alive_s));

 ret = iv_get_keep_alive_info(&sg_keep_alive_cfg.keep_alive_info,
timeout_ms);
 if (ret) {
     Log_e("get keep alive info failed %d", ret);
     return ret;
 }
 sg_keep_alive_cfg.running = 1;
 return ret;
}
```

2. 模拟设备保活。

```
int simulation_device_low_power_keep_alive(void)
{
 static uintptr_t _keepalive_fd     = 0;
 iv_keep_alive_info_s *p_keep_alive =
&sg_keep_alive_cfg.keep_alive_info;
 int rc                             = 0;
 uint8_t _tmp_msg[256];
```

```
 sg_keep_alive_cfg.start = 1;
 while (sg_keep_alive_cfg.running) {
     if (_keepalive_fd == 0) {
         Log_d("fd invalid, try to connect...");
         _keepalive_fd =
             _connect_auth_keepalive(p_keep_alive->server_addr,
p_keep_alive->port,
                                     p_keep_alive->auth_msg,
p_keep_alive->auth_msg_len);
         usleep(5 * 1000);
         continue;
     }
     size_t _recv_len = 0;
     rc = qcloud_iv_tcp_read(_keepalive_fd, _tmp_msg, p_keep_alive-
>wake_up_msg_len, 60000,
                             &_recv_len);
     if (rc == QCLOUD_IV_ERR_TCP_PEER_SHUTDOWN) {
         Log_d("connection lost");
         qcloud_iv_disconnect(_keepalive_fd);
         _keepalive_fd = 0;
         continue;
     }
     if (_recv_len > 0) {
         Log_d("Recv message from server");
         _dump_hex_msg(_tmp_msg, _recv_len);
         if (_recv_len == p_keep_alive->wake_up_msg_len &&
             !memcmp(_tmp_msg, p_keep_alive->wake_up_msg,
p_keep_alive->wake_up_msg_len)) {
             Log_d("Recv wakeup message....");
             break;
         }
     }

     Log_d("Sending keepalive message");
     size_t _send_len = 0;
     rc               = qcloud_iv_tcp_write(_keepalive_fd,
p_keep_alive->heart_beat_msg,
                            p_keep_alive->heart_beat_msg_len, 100,
&_send_len);
     if (_send_len != p_keep_alive->heart_beat_msg_len) {
         Log_e("tcp write %d %u", rc, _send_len);
     }
```

```
        sleep(5);
    }
    sg_keep_alive_cfg.start = 0;
    return 0;
}
```

OTA 模块

最近更新时间：2024-09-06 16:43:51

功能介绍

本模块用于将设备端实现 OTA 功能，当设备有新功能或者需要修复漏洞时，设备可以通过 OTA 服务快速的进行固件升级。

使用流程

固件升级的过程中，通过 MQTT 实现与云端的信令交互，上报本地固件版本号，等待云端下发升级指令，并上报升级状态。在拿到固件的 URL 地址之后，会通过 HTTP 下载固件。

> **注意：**
> SDK 会在用户提供的固件下载目录下面创建包含固件版本号的 **.bin** 文件，某些平台对文件名的长度有限制，因此在云端控制台创建固件版本号不宜太长。

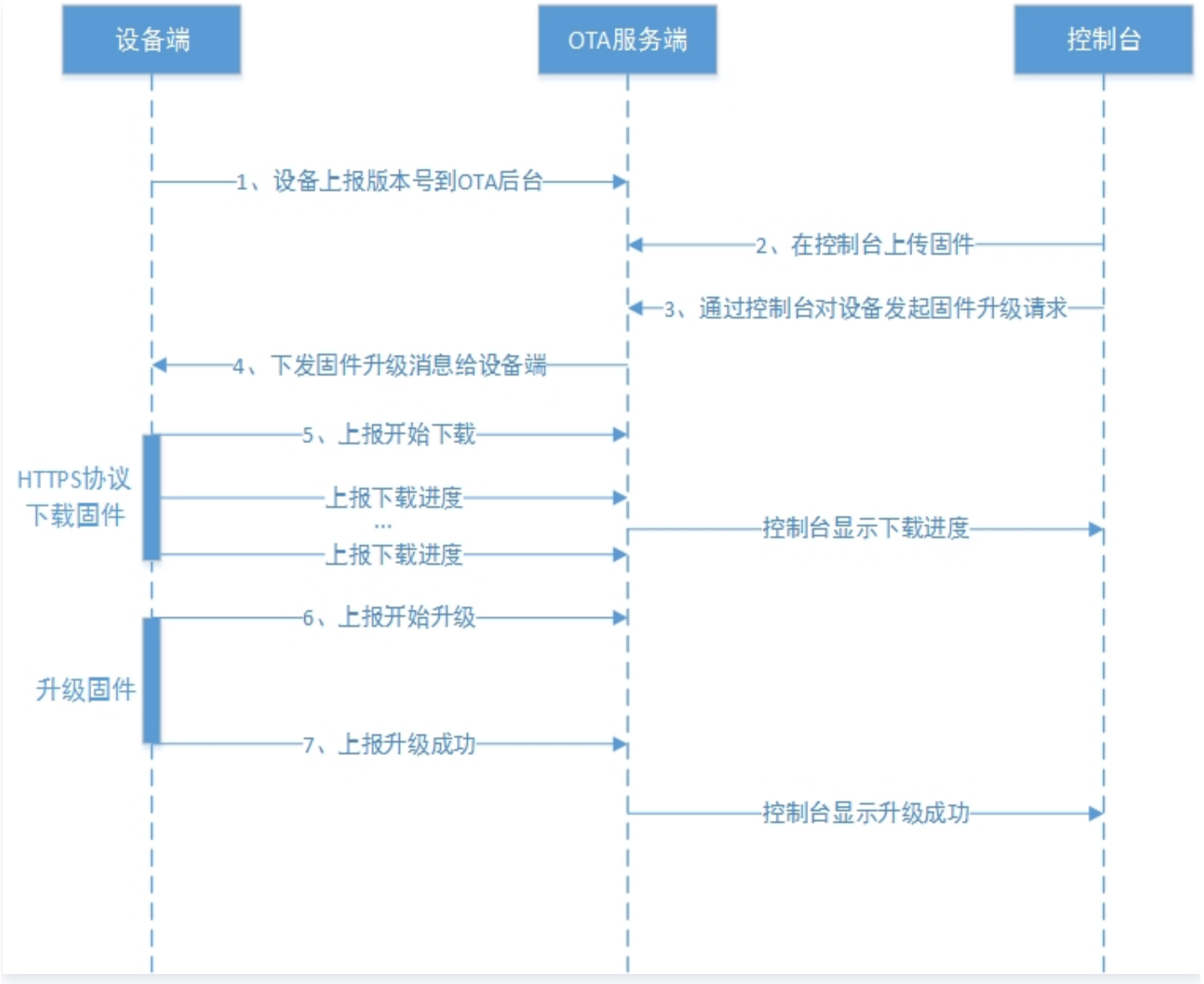
```
iv_ota_init            //提供本地固件版本及下载路径，开始OTA流程
    |
iv_ota_firmware_update_cb // 固件下载完成之后，收到回调，开始更新固件
    |
iv_ota_update_progress // 更新固件过程中通过该函数上报结果
    |
iv_ota_exit            // 更新成功或出现异常，都需要停止OTA模块运行
```

接口简介

接口函数声明请参考 **iv_ota.h**，具体内容请 单击此处 进行 SDK 获取。

操作流程

设备的升级流程如下所示：



1. 设备上报当前版本号。
   当执行 iv_ota_init 之后，SDK 会启动单独的 OTA 任务线程，并通过 MQTT 协议进行版本号的上报，消息为 json 格式，内容如下：

```
{
 "type": "report_version",
 "report":{
     "version": "0.1"
 }
}
// type：消息类型
// version：上报的版本号
```

2. 然后您可以在控制台上传固件。
   具体控制台操作指引，可以参考 固件升级 。
3. 在控制台将指定的设备升级到指定的版本。
4. 触发固件升级操作后，设备端 SDK 会收到固件升级的消息，内容如下：

```
{
 "file_size": 708482,
 "md5sum": "36eb59511****14a631463a37a9322a2",
 "type": "update_firmware",
 "url": "https://ota-1255858890.cos.ap-guangzhou.myqcloud.com",
 "version": "0.2"
}
// type：消息类型为update_firmware
// version：升级版本
// url：下载固件的url
// md5asum：固件的MD5值
// file_size：固件大小，单位为字节
```

5. 设备端 SDK 在收到固件升级的消息后，会根据 URL 下载固件，下载的过程中设备 SDK 会不断的上报下载进度，上报的内容如下：

```
{
 "type": "report_progress",
 "report":{
     "progress":{
            "state":"downloading",
              "percent":"10",
            "result_code":"0",
            "result_msg":""
     },
     "version": "0.2"
 }
}
// type：消息类型
// state：状态为正在下载中
// percent：当前下载进度，百分比
```

6. 当设备端 SDK 下载完固件并校验 MD5 正确之后，会通过回调 iv_ota_firmware_update_cb 通知使用者，并告知固件地址及相关信息。
   使用者在开始更新固件之前，需要通过 iv_ota_update_progress 来触发 SDK 上报一条开始升级的消息，内容如下：

```
{
    "type": "report_progress",
    "report":{
        "progress":{
                "state":"burning",
                "result_code":"0",
                "result_msg":""
        },
        "version": "0.2"
    }
}
// type：消息类型
// state：状态为烧制中
```

7. 设备固件升级完成后，再通过 iv_ota_update_progress 来触发 SDK 上报升级成功消息，内容如下：

```
{
 "type": "report_progress",
 "report":{
     "progress":{
             "state":"done",
             "result_code":"0",
             "result_msg":""
     },
     "version": "0.2"
 }
}
// type：消息类型
// state：状态为已完成
```

> **注意：**
> 在下载固件或升级固件的过程中，如果失败，同样通过 iv_ota_update_progress 来触发 SDK 上报升级失败消息，内容如下：

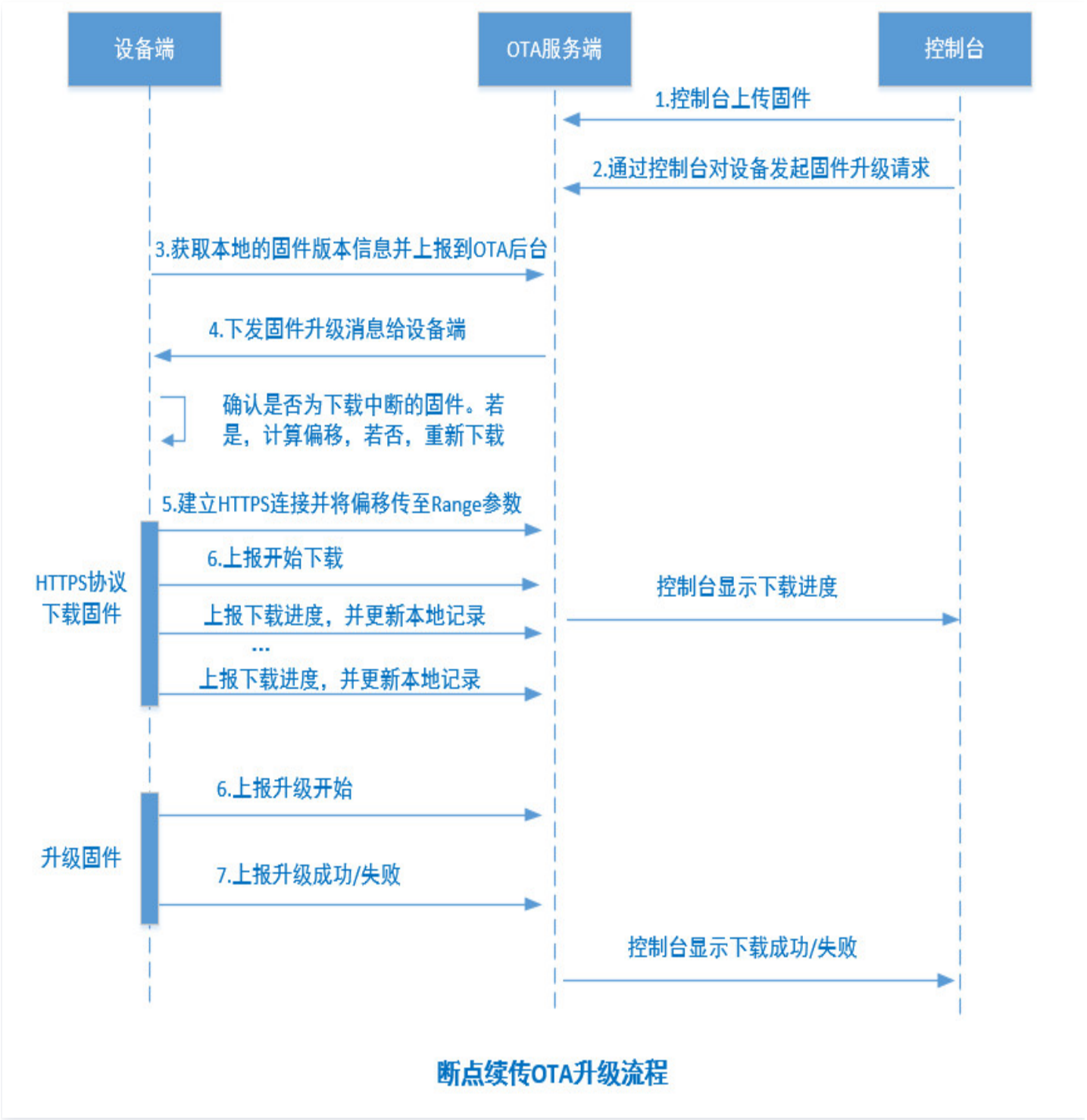
```
{
 "type": "report_progress",
 "report":{
     "progress":{
             "state":"fail",
             "result_code":"-1",
```

```
            "result_msg":"time_out"
        },
        "version": "0.2"
    }
}
// state：状态为失败
// result_code：错误码，-1：下载超时；-2：文件不存在；-3：签名过期；-4:MD5不匹配；-5：更新固件失败
// result_msg：错误消息
```

8. 当固件升级结束之后，或者发生异常需要退出的时候，需要调用 **iv_ota_exit** 退出 OTA 模块，否则会循环上报版本并进行 OTA 过程。

OTA断点续传

物联网设备有部分场景处于弱网环境，在这个场景下连接会不稳定，固件下载会中断的情况出现。如果每次都从0偏移开始下载固件，则弱网环境有可能一直无法完成全部固件下载，因此固件的断点续传功能特别必要。
断点续传就是从文件上次中断的地方开始重新下载或上传，要实现断点续传的功能，需要设备端记录固件下载的中断位置，同时记录下载固件的 md5、文件大小、版本信息。
平台针对 OTA 中断的场景，设备侧 report 设备的版本。如果上报的版本号与目标升级的版本号不一致，则平台会再次下发固件升级消息。设备接收到目标固件信息后，将其与本地记录的中断固件信息进行比较。确认为同一固件后，设备将从断点处继续下载。
目前 SDK 会在用户提供的固件下载路径下，创建 json 文件实现记录描述，如果固件下载路径下除了 bin 固件文件，还存在一个文本文件，则说明固件下载还未完成，在固件下载完成并校验无误之后，SDK 会自动删除文本描述文件。
带断点续传的 OTA 升级流程如下，弱网环境下第3步到第6步有可能会多次执行，没有执行第7步，执行第3步，设备端都会收到第4步的消息。



断点续传OTA升级流程

公共数据结构

最近更新时间：2024-08-23 17:43:51

功能介绍

SDK 提供若干基础数据结构，供用户推送音视频流使用。

枚举说明

iv_cm_aenc_type_e

用于描述音频编码类型。

枚举原型

```
typedef enum
{
    IV_CM_AENC_TYPE_PCM    = 0,
    IV_CM_AENC_TYPE_G711A  = 1,
    IV_CM_AENC_TYPE_G711U  = 2,
    IV_CM_AENC_TYPE_G726   = 3, // 暂不支持
    IV_CM_AENC_TYPE_AAC    = 4,
    IV_CM_AENC_TYPE_AMR    = 5, // 暂不支持
    IV_CM_AENC_TYPE_ADPCMA = 6, // 暂不支持

    IV_CM_AENC_TYPE_BUTT
} iv_cm_aenc_type_e;
```

iv_cm_aac_type_e

用于描述 AAC 音频编码类型。

枚举原型

```
typedef enum
{
    IV_CM_AAC_TYPE_NONE      = 0,
    IV_CM_AAC_TYPE_MAIN      = 1,
    IV_CM_AAC_TYPE_LC        = 2,
    IV_CM_AAC_TYPE_SSR       = 3,
    IV_CM_AAC_TYPE_LTP       = 4,
    IV_CM_AAC_TYPE_SBR       = 5,
    IV_CM_AAC_TYPE_SCALCBLE  = 6,
    IV_CM_AAC_TYPE_TWINVQ    = 7,
```

```
    IV_CM_AAC_TYPE_CELP      = 8,
    IV_CM_AAC_TYPE_HVXC      = 9,

    IV_CM_AAC_TYPE_BUTT
} iv_cm_aac_type_e;
```

iv_cm_aenc_sample_rate_e

用于描述音频采样率，仅 PCM 格式使用，其他格式无效。

枚举原型

```
typedef enum
{
    IV_CM_AENC_SAMPLE_RATE_8000  = 8000,  /* 8K samplerate*/
    IV_CM_AENC_SAMPLE_RATE_12000 = 12000, /* 12K samplerate*/
    IV_CM_AENC_SAMPLE_RATE_11025 = 11025, /* 11.025K samplerate*/
    IV_CM_AENC_SAMPLE_RATE_16000 = 16000, /* 16K samplerate*/
    IV_CM_AENC_SAMPLE_RATE_22050 = 22050, /* 22.050K samplerate*/
    IV_CM_AENC_SAMPLE_RATE_24000 = 24000, /* 24K samplerate*/
    IV_CM_AENC_SAMPLE_RATE_32000 = 32000, /* 32K samplerate*/
    IV_CM_AENC_SAMPLE_RATE_44100 = 44100, /* 44.1K samplerate*/
    IV_CM_AENC_SAMPLE_RATE_48000 = 48000, /* 48K samplerate*/
    IV_CM_AENC_SAMPLE_RATE_64000 = 64000, /* 64K samplerate*/
    IV_CM_AENC_SAMPLE_RATE_96000 = 96000, /* 96K samplerate*/

    IV_CM_AENC_SAMPLE_RATE_BUTT,
} iv_cm_aenc_sample_rate_e;
```

iv_cm_aenc_bit_width_e

用于描述音频位宽，仅 PCM 格式使用，其他格式无效。

枚举原型

```
typedef enum
{
    IV_CM_AENC_BIT_WIDTH_8  = 0, /* 8bit width */
    IV_CM_AENC_BIT_WIDTH_16 = 1, /* 16bit width*/
    IV_CM_AENC_BIT_WIDTH_24 = 2, /* 24bit width*/

    IV_CM_AENC_BIT_WIDTH_BUTT,
} iv_cm_aenc_bit_width_e;
```

iv_cm_aenc_mode_e

用于描述音频声道数，仅 G711a、G711u、PCM 使用，其他格式无效。

枚举原型

```
typedef enum
{
    IV_CM_AENC_MODE_MONO   = 0, /*mono*/
    IV_CM_AENC_MODE_STEREO = 1, /*stereo*/

    IV_CM_AENC_MODE_BUTT
} iv_cm_aenc_mode_e;
```

iv_cm_file_type_e

用于描述文件类型,用户可自行扩展。

枚举原型

```
typedef enum
{
    IV_CM_FILE_TYPE_VIDEO = 0, /*video file*/
    IV_CM_FILE_TYPE_PIC   = 1  /*pircture file*/

    /*User expandable*/
} iv_cm_file_type_e;
```

iv_cm_stream_type_e

用于描述媒体流的类型。

枚举原型

```
typedef enum
{
    IV_CM_STREAM_TYPE_AUDIO = 0,
    IV_CM_STREAM_TYPE_VIDEO = 1,

    IV_CM_STREAM_TYPE_BUTT
} iv_cm_stream_type_e;
```

iv_cm_venc_type_e

用于描述视频流的编码格式。

枚举原型

```
typedef enum
{
    IV_CM_VENC_TYPE_H264 = 0, /*h264*/
    IV_CM_VENC_TYPE_H265 = 1, /*h265*/

    IV_CM_VENC_TYPE_BUTT
} iv_cm_venc_type_e;
```

iv_cm_frame_type_e

用于描述视频帧的类型，暂不支持 B帧。
枚举原型

```
typedef enum
{
    IV_CM_FRAME_TYPE_I = 0,
    IV_CM_FRAME_TYPE_P = 1,
    // IV_CM_FRAME_TYPE_B = 2, // not support yet

    IV_CM_FRAME_TYPE_BUTT
} iv_cm_frame_type_e;
```

iv_cm_wifi_enc_type_e

用于描述 wifi 加密类型。
枚举原型

```
typedef enum
{
    IV_CM_WIFI_ENC_TYPE_OPEN = 0, /*open wifi*/
    IV_CM_WIFI_ENC_TYPE_WEP  = 1, /*wep wifi*/
    IV_CM_WIFI_ENC_TYPE_WPA  = 2, /*wpa wifi*/

    IV_CM_WIFI_ENC_TYPE_BUTT
} iv_cm_wifi_enc_type_e;
```

数据结构说明

iv_cm_aenc_pack_s

音频帧数据包。

结构原型

```
typedef struct iv_cm_aenc_pack_s {
    uint8_t *pu8Addr;
    uint32_t u32Len;
    uint64_t u64PTS;
    uint32_t u32Seq;
} iv_cm_aenc_pack_s;
```

参数说明

成员名称	描述	取值
pu8Addr	音频帧数据的地址。	uint8_t *
u32Len	音频帧数据长度（字节）。	uint32_t
u64PTS	音频帧数据时间戳，采用相对时间，可从0或 UTC 时间等任意时间基准开始计时，单位毫秒。	uint64_t
u32Seq	音频帧数据包序号，每包+1。	uint32_t

iv_cm_aenc_stream_s

音频数据流，由1-8个音频帧数据包组成。

结构原型

```
typedef struct iv_cm_aenc_stream_s {
    iv_cm_aenc_pack_s *pstAencPack[8];
    uint32_t u32PackCount;
} iv_cm_aenc_stream_s;
```

参数说明

成员名称	描述	取值
pstAencPack	音频帧数据包的指针数组。	iv_cm_aenc_pack_s *
u32PackCount	音频帧数据包个数。	uint32_t

iv_cm_venc_pack_s

视频帧数据包。

结构原型

```
typedef struct iv_cm_venc_pack_s {
    uint8_t *pu8Addr;
    uint32_t u32Len;
    uint64_t u64PTS;
    iv_cm_frame_type_e eFrameType;
    uint32_t u32Seq;
} iv_cm_venc_pack_s;
```

参数说明

成员名称	描述	取值
pu8Addr	视频帧数据的地址。	uint8_t *
u32Len	视频帧数据长度（字节）。	uint32_t
u64PTS	视频帧数据时间戳，采用相对时间，可从0或 UTC 时间等任意时间基准开始计时，单位毫秒。	uint64_t
eFrameType	视频帧类型。	iv_cm_frame_type_e
u32Seq	视频帧数据包序号，每包+1。	uint32_t

iv_cm_venc_stream_s

视频数据流，由1-8个视频帧数据包组成。

结构原型

```
typedef struct iv_cm_venc_stream_s {
    iv_cm_venc_pack_s *pstVencPack[8];
    uint32_t u32PackCount;
} iv_cm_venc_stream_s;
```

参数说明

成员名称	描述	取值
pstVencPack	视频帧数据包的指针数组。	iv_cm_venc_pack_s *
u32PackCount	视频帧数据包个数。	uint32_t

iv_cm_avenc_stream_s

音视频数据流，由一个音频流和一个视频流组成。
结构原型

```
typedef struct iv_cm_avenc_stream_s {
    iv_cm_aenc_stream_s stAencStream;
    iv_cm_venc_stream_s stVencStream;
} iv_cm_avenc_stream_s;
```

参数说明

成员名称	描述	取值
stAencStream	音频流	iv_cm_aenc_stream_s
stVencStream	视频流	iv_cm_venc_stream_s

iv_cm_av_data_info_s

用于描述音视频格式。
结构原型

```
typedef struct iv_cm_av_data_info_s {
    iv_cm_aenc_type_e eAudioType;
    iv_cm_aac_type_e u32AudioCodecOption;
    iv_cm_aenc_mode_e eAudioMode;
    iv_cm_aenc_bit_width_e eAudioBitWidth;
    iv_cm_aenc_sample_rate_e eAudioSampleRate;
    uint32_t u32SampleNumPerFrame;

    iv_cm_venc_type_e eVideoType;
    uint32_t u32VideoWidth;
    uint32_t u32VideoHeight;
    uint32_t u32Framerate;

    uint32_t u32Reserve[4];
```

```
} iv_cm_av_data_info_s;
```

参数说明

成员名称	描述	取值
eAudioType	音频编码类型。	iv_cm_aenc_type_e
u32AudioCodecOption	aac 编码类型，仅 AAC 使用，其他格式无效。	iv_cm_aac_type_e
eAudioMode	音频声道数，仅 G711a、G711u、PCM 使用，其他格式无效。	iv_cm_aenc_mode_e
eAudioBitWidth	音频位宽，仅 PCM 使用，其他格式无效。	iv_cm_aenc_bit_width_e
eAudioSampleRate	音频采样率，仅 PCM 使用，其他格式无效。	iv_cm_aenc_sample_rate_e
u32SampleNumPerFrame	音频每帧采样点数，仅 G711a、G711u、PCM 使用，其他格式无效（采样点数需乘以声道数，例如单声道1024点；双声道共2048点，即每声道1024点）。	uint32_t
eVideoType	视频编码类型。	iv_cm_venc_type_e
u32VideoWidth	视频宽度。	uint32_t
u32VideoHeight	视频高度。	uint32_t
u32Framerate	视频帧率，仅支持整数帧率。	uint32_t
u32Reserve	保留。	uint32_t

iv_cm_memory_s

内存管理。

结构原型

```
typedef struct {
    uint8_t *buf;                              // buffer address
    size_t size;                               // buffer size
    void (*buf_free_fn)(uint8_t *, size_t);    // buffer free callback
} iv_cm_memory_s;
```

参数说明

成员名称	描述	取值
buf	内存地址。	uint8_t *
size	内存大小。	size_t
buf_free_fn	内存释放回调函数。	void (*)(uint8_t *, size_t)

iv_cm_time_fragment_s

时间段。

结构原型

```
typedef struct {
    uint32_t type;           // not used
    uint64_t begin_time_s;   // start timestamp in second of playback,
unix timestampe
    uint64_t end_time_s;     // end timestamp in second of playback, unix
timestampe
} iv_cm_time_fragment_s;
```

参数说明

成员名称	描述	取值
type	类型，暂时未使用。	uint32_t
begin_time_s	起始时间。	单位秒，UNIX 时间戳。
end_time_s	结束时间。	单位秒，UNIX 时间戳。

iv_cm_pb_list_s

回放的录像列表。

结构原型

```
typedef struct {
    uint32_t count;                        // file count of the day
    iv_cm_time_fragment_s file_list[0];    // play back file array
} iv_cm_pb_list_s;
```

参数说明

成员名称	描述	取值
count	录像的数量。	uint32_t
file_list	录像实际时间段。	iv_cm_time_fragment_s

iv_cm_query_rd_by_month_s

按月查询录像结果。

结构原型

```
typedef struct {
    uint32_t year;
    uint32_t month;
    uint32_t type;  // not used
    uint32_t day;   // Use each bit for one day. For example day=0010
0000 1000 0000=8320 means
                    // 8/14 day have playback data.
} iv_cm_query_rd_by_month_s;
```

参数说明

成员名称	描述	取值
year	年	uint32_t
month	月	1到12
type	类型，暂时未使用。	uint32_t
day	有录像的天数,从低到高 bit 依次代表第几天。	uint32_t

iv_cm_query_rd_by_day_s

按月查询录像结果。

结构原型

```
typedef struct {
    iv_cm_time_fragment_s time_fragment;
    void (*free_fn)(void *ptr);  // memory free callback
    iv_cm_pb_list_s *rd_array;
} iv_cm_query_rd_by_day_s;
```

参数说明

成员名称	描述	取值
iv_cm_time_fragment_s	查询时间段。	iv_cm_time_fragment_s
free_fn	查询结果释放函数。	函数指针
rd_array	查询结果。	iv_cm_pb_list_s *

iv_cm_file_info_s

文件信息。

结构原型

```
typedef struct {
    iv_cm_file_type_e file_type;  // file type
    char file_name[MAX_FILE_NAME_LENGTH];          // file name
    uint32_t file_size;           // file size, unit Byte
    uint32_t begin_time_s;        // start timestamp in second of
playback, unix timestampe
    uint32_t end_time_s;          // end timestamp in second of
playback, unix timestampe
    iv_cm_memory_s extra_info;    // user define information, json
format
} iv_cm_file_info_s;
```

参数说明

成员名称	描述	取值
file_type	文件类型。	iv_cm_file_type_e
file_name	文件名称。	字符串
file_size	文件大小。	uint32_t
begin_time_s	文件起始时间。	uint32_t
end_time_s	文件结束时间。	uint32_t
extra_info	文件自定义信息，json 格式。	iv_cm_memory_s

iv_cm_file_list_s

文件列表。

结构原型

```
typedef struct {
    uint32_t count;                    // file count
    iv_cm_file_info_s file_list[0];   // file list
} iv_cm_file_list_s;
```

参数说明

成员名称	描述	取值
count	文件数量	uint32_t
iv_cm_file_info_s	文件信息	iv_cm_file_info_s

iv_cm_file_list_s

文件列表查询参数。
结构原型

```
typedef struct {
    iv_cm_time_fragment_s time_fragment;
    void (*free_fn)(void *ptr);   // memory free callback
    iv_cm_file_list_s *file_array;
} iv_cm_query_file_list_s;
```

参数说明

成员名称	描述	取值
count	文件数量	uint32_t
iv_cm_file_info_s	文件信息	iv_cm_file_info_s

iv_cm_download_param_s

文件列表查询参数。
结构原型

```
typedef struct {
    char file_name[MAX_FILE_NAME_LENGTH];  // name of downloaded file
    int file_offset;      // offset of downloaded file
} iv_cm_download_param_s;
```

参数说明

成员名称	描述	取值
file_name	文件名	char[]
file_offset	文件偏移	int

iv_cm_pb_seek_s

回放时滑动位置。

结构原型

```
typedef struct {
    uint64_t seek_time_ms;
} iv_cm_pb_seek_s;
```

参数说明

成员名称	描述	取值
seek_time_ms	UNIX 时间戳，单位毫秒。	uint64_t

iv_cm_pb_ff_s

设置快进速度。

结构原型

```
typedef struct {
    uint32_t speed;
} iv_cm_pb_ff_s;
```

参数说明

成员名称	描述	取值
speed	• 0：正常播放 。 • 1：表示只发I帧。 • 2：表示I帧取2发一。 • 3：表示I帧取3发1。	uint32_t

iv_cm_pb_speed_s

设置快放或者慢放。

结构原型

```
typedef struct {
    uint32_t time_ms;
} iv_cm_pb_speed_s;
```

参数说明

成员名称	描述	取值
time_ms	表示 pts 的间隔，当大于正常 pts 时慢放，否则是快放。	uint32_t

iv_cm_pb_rewind_s

设置倒放。

结构原型

```
typedef struct {
    uint64_t begin_time_s;  // end timestamp of rewind
    uint64_t end_time_s;    // start timestamp of rewind
} iv_cm_pb_rewind_s;
```

参数说明

成员名称	描述	取值
begin_time_s	倒放的截止时间，UNIX 时间戳。	uint64_t
end_time_s	倒放的起始时间，UNIX 时间戳。	uint64_t

注意事项

- 对于音视频数据的时间戳，请小心处理误差问题，否则可能因误差积累导致音视频不同步。
  对于视频帧，假设帧率为30fps，即每帧33.3333毫秒，取整为33毫秒。不补偿时间戳为 0,33,66,99,132,165,198，补偿后时间戳为 0,33,67,100,133,167,200。
  音频同理。
- 音频和视频的时间戳必须同时从0或同时从 UTC 时间等相同时间基准开始计时，不得一个从0计时，另一个从 UTC 计时，否则会导致音频无法播放等问题。
- 录像快进、倒放、快放和慢放状态时，音频需要特殊处理，可以不发送。

错误码

最近更新时间：2024-08-23 17:43:51

简介

SDK 的错误码说明，参考头文件 "iv_err.h"。

说明

错误码	取值	说明
IV_ERR_NONE	0	成功。

系统模块错误码

系统模块错误码	取值	说明
IV_ERR_SYS_INIT_PRM_NULL	-100	系统模块初始化参数为空。
IV_ERR_SYS_INIT_CB_NULL	-101	系统模块初始化回调函数为空。
IV_ERR_SYS_INIT_PRM_RANGE	-102	系统模块初始化参数超过范围。
IV_ERR_SYS_DEVICE_INFORMATION	-103	系统模块获取设备信息错误。
IV_ERR_SYS_DYNAMIC_REG_DEVIECE	-104	动态注册设备失败。
IV_ERR_SYS_NOT_SUPPORT	-105	暂不支持此功能，此功能被裁剪，此功能已淘汰等。
IV_ERR_SYS_NTP_NOT_AVAILABLE	-106	NTP 时间错误，NTP 时间不可用，NTP 时间不精确等。

物模型模块错误码

物模型模块错误码	取值	说明
IV_ERR_DM_INIT_PRM_NULL	-200	物模型模块初始化参数为空。

IV_ERR_DM_INIT_CB_NULL	-201	物模型模块初始化回调函数为空。
IV_ERR_DM_INIT_PRM_RANGE	-202	物模型模块参数超过范围。
IV_ERR_DM_INIT_ENV	-203	物模型模块初始化环境错误。
IV_ERR_DM_TYPE_NOT_SUPPORT	-204	物模型模块类型不支持。
IV_ERR_DM_NULL_PTR	-205	物模型输入参数空指针。
IV_ERR_DM_REPORT_EVENT_FAIL	-206	事件上报失败。
IV_ERR_DM_REPORT_BUSY	-207	属性上报忙。
IV_ERR_DM_REQUEST_BUSY	-208	属性请求忙。

音视频对讲模块错误码

音视频对讲模块错误码	取值	说明
IV_ERR_AVT_INIT_PRM_NULL	-300	音视频传输和对讲模块初始化参数为空。
IV_ERR_AVT_INIT_CB_NULL	-301	音视频传输和对讲模块初始化回调函数为空。
IV_ERR_AVT_INIT_PRM_RANGE	-302	音视频传输和对讲模块参数超过范围。
IV_ERR_AVT_REQ_CHN_BUSY	-303	音视频传输和对讲模块请求通道忙。
IV_ERR_AVT_SEND_STREAM_TOO_BIG	-304	发送的数据超过初始设置的最大值。
IV_ERR_AVT_CHN_NOT_EXIT	-305	请求的通道不存在。
IV_ERR_AVT_NEED_IDR_FRAME	-306	需要关键帧。
IV_ERR_AVT_MALLOC_BUFFER_FAILED	-307	分配的内存失败。

IV_ERR_AVT_FAILED	-308	音视频传输和对讲模块运行错误。
IV_ERR_AVT_INPUT_PARAM_NULL	-309	接口入参为空。
IV_ERR_AVT_INPUT_PARAM_INVAILD	-310	接口入参无效。
IV_ERR_AVT_SEND_DATA_TIMEOUT	-311	接口调用超时。

配网模块错误码

配网模块错误码	取值	说明
IV_ERR_AD_QR_NO_PARSE_RESULT	-400	配网模块解析二维码失败。
IV_ERR_AD_QR_PARSE_PRM_RANGE	-401	配网模块解析二维码参数超出范围。
IV_ERR_AD_INIT_PRM_RANGE	-402	配网模块初始化参数超过范围。
IV_ERR_AD_INIT_PRM_NULL	-403	配网模块初始化参数为空。
IV_ERR_AD_INIT_CB_NULL	-404	配网模块初始化回调函数为空。
IV_ERR_AD_SUBSCRIBE_FAIL	-405	配网模块订阅消息失败。
IV_ERR_AD_PUBLISH_FAIL	-406	配网模块发布消息失败。
IV_ERR_AD_TIME_OUT	-407	配网超时。

本地录像模块错误码

本地录像模块错误码	取值	说明
IV_ERR_RD_INIT_PRM_NULL	-500	本地录像模块初始化参数为空。
IV_ERR_RD_INIT_CB_NULL	-501	本地录像模块初始化回调为空。
IV_ERR_RD_INIT_PRM_RANGE	-502	本地录像模块初始化超过范围。

IV_ERR_RD_INIT_SD_PATH_INVALID	-503	本地录像模块 SD 路径无效。
IV_ERR_RD_CREATE_RECORD_DIR	-504	本地录像模块创建存储目录错误。

云存模块错误码

云存模块错误码	取值	说明
IV_ERR_CS_INIT_PRM_NULL	-600	云存模块初始化参数为 NULL。
IV_ERR_CS_INIT_CB_NULL	-601	云存模块初始化回调函数为空。
IV_ERR_CS_INIT_PRM_RANGE	-602	云存模块输入参数超过范围。
IV_ERR_CS_UPLOAD_AUTH_NOT_AVAILABLE	-603	云存模块未开通权限。
IV_ERR_CS_UPLOAD_OPEN_FILE_FAIL	-604	云存模块打开缓存文件失败。
IV_ERR_CS_PRM_NOT_AVAILABLE	-605	参数不可用，未初始化。
IV_ERR_CS_PRM_MALLOC_FAIL	-606	云存模块分配内存失败。
IV_ERR_CS_APPLY_NO_SERVICE	-607	云存套餐不可用。
IV_ERR_CS_QUERY_SERVICE_TIMEOUT	-608	查询云存套餐超时。
IV_ERR_CS_EVENT_IS_VALID	-609	云存事件无效。
IV_ERR_CS_INIT_REPEAT	-610	云存重复初始化。

自定义信令模块错误码

自定义信令模块错误码	取值	说明
IV_ERR_UC_INIT_PRM_NULL	-700	自定义信令模块初始化参数为 NULL。
IV_ERR_UC_PRM_RANGE	-701	自定义信令模块接口参数错误。

IV_ERR_UC_INIT_CB_NULL	-702	自定义信令模块初始化回调函数为空。
IV_ERR_UC_INIT_PRO_FAIL	-703	自定义信令模块初始化过程中失败。
IV_ERR_UC_MSG_SEND_FAIL	-704	自定义信令模块数据发送失败。
IV_ERR_UC_MSG_LEN_RANGE	-705	自定义信令模块数据长度超出范围。

云 AI 模块错误码

云 AI 模块错误码	取值	说明
IV_ERR_CLOUDAI_WRONG_PARA	-750	云 AI 模块参数错误。
IV_ERR_CLOUDAI_HTTP_REQ	-751	云 AI HTTP 请求错误。
IV_ERR_CLOUDAI_STOP	-752	云 AI 模块业务停止。
IV_ERR_CLOUDAI_INIT_FAIL	-753	云 AI 模块参数错误。
IV_ERR_CLOUDAI_FILE_FAIL	-754	云 AI 模块文件操作失败。
IV_ERR_CLOUDAI_PROPERTY_FAIL	-755	云 AI 模块物模型操作失败。
IV_ERR_CLOUDAI_MQTT_PUBILSH	-756	云 AI 模块 mqtt 消息发布失败。
IV_ERR_CLOUDAI_MQTT_TIMEOUT	-757	云 AI 模块 mqtt 消息超时。
IV_ERR_CLOUDAI_MEMORY_FAIL	-758	云 AI 模块分配内存失败。
IV_ERR_CLOUDAI_MODEL_FAIL	-759	云 AI 模块模型 ID 错误。
IV_ERR_CLOUDAI_COMM_FAIL	-760	云 AI 模块内部通用错误。

OTA模块错误码

OTA 模块错误码	取值	说明

IV_ERR_OTA_INIT_PRM_NULL	-800	OTA 模块初始化参数为空。
IV_ERR_OTA_INIT_CB_NULL	-801	OTA 模块回调函数为空。
IV_ERR_OTA_START_FAIL	-802	无法启动 OTA 升级。
IV_ERR_OTA_PROGRESS_TYPE_ERROR	-803	OTA 升级失败。

低功耗保活模块错误码

低功耗保活模块错误码	取值	说明
IV_ERR_KP_INPUT_PRM_NULL	-900	低功耗保活模块初始化参数为空。
IV_ERR_KP_REGISTER_FAILED	-901	低功耗保活模块注册失败。
IV_ERR_KP_SEND_FAILED	-902	低功耗保活模块发送消息失败。
IV_ERR_KP_GET_TIMEOUT	-903	低功耗保活模块等待响应超时。

其他错误码

其他错误码	取值	说明
IV_ERR_DEVICE_OFFLINE	-1001	设备处于离线状态。

常见问题解答

最近更新时间：2024-09-06 15:42:31

云存相关问题

云存录像回放时间异常

详细描述

云存录像回放时间异常，包括但不限于以下情况：

- 音视频不同步。
- 前30秒只有音频，从第30秒开始有音频和视频，音视频之间相差30秒（或类似问题）。
- 视频回放时播放速度时快时慢。
- 实际录像时长为1分钟，回放时画面只显示了一瞬间，播放器闪退。
- 实际录像时长为1分钟，播放器进度条显示时长为16小时，且画面卡住不动。
- 实际录像时长为1分钟，播放器进度条显示时长远超1分钟，且无画面或无声音或即无画面也无声音。
  部分播放器对于这类情况的处理也不同，最终实际的播放效果也不同，上述现象仅供参考。

原因分析

以上几种问题或类似问题都是音视频帧时间戳异常导致的。云存视频回放严格依赖时间戳，因此在推送音视频帧时务必保证时间戳正确。
下面进行逐个分析：

- **音视频不同步**

  通常来说音视频帧的时间戳是这一帧采集的时刻，一般硬件编码器都带有时间戳，这个时间戳建议直接从编码器取出。如果编码器不带时间戳在手动添加时间戳时请尽量保证时间戳准确。
  手动添加时间戳常见错误是没有考虑误差积累，特别是音频有重采样、格式转换等操作，视频有改变帧率等操作更容易引入误差，导致音视频帧的时间戳误差越来越大。

  > 例如视频帧率是 30fps，每帧之间相差33.333……毫秒，整除为33毫秒，不进行误差补偿的时间戳为0,33,66,99,132,165,198，补偿后为 0,33,66,100,133,166,200。

  另一种情形是硬件编码的时间戳从初始化的那一刻开始计时，例如音频编码器在第0秒初始化，视频编码器在第3秒初始化，两个编码器的时间戳都是从0开始计时，两个编码器的时间戳虽然都从0开始计时，但因为初始化时间不同使拿到的音视频时间戳就始终相差3秒，导致音视频不同步。
- **前30秒只有音频，从第30秒开始有音频和视频，音视频之间相差30秒（或类似问题）。**

  同样属于音视频不同步问题，见上文。
- **视频回放时播放速度时快时慢。**

  为了光线不足时的画质，部分芯片在夜间或黑暗环境下会延长曝光时间来提升画面亮度，因此导致帧率降低。这种情况往往是手动计算帧率导致的。

例如明亮环境下默认20fps，黑暗环境下降低为10fps，但时间戳仍然按20fps计算。
如下所示，假设在第5帧处帧率由20fps变为10fps：
- 正常时间戳：0,50,100,150,200,300,400,500,600,700
- 异常时间戳：0,50,100,150,200,250,300,350,400,450

最终的效果就是明亮环境下画面正常，黑暗环境下画面速度为正常的2倍。如果芯片的 ISP 算法会改变帧率，建议直接从编码器获取时间戳，如需手动计算时间戳请按实际帧率计算，或者在向 SDK 推送视频帧时直接从系统获取毫秒级时间戳。

- **实际录像时长为1分钟，回放时画面只显示了一瞬间，播放器闪退。**

  音视频帧的时间戳填写错误，典型情况就是误将时间戳填写为帧序号。
  假设帧率为20fps，正常时间戳：0,50,100,150,200,300,400,500,600,700，异常时间戳：0,1,2,3,4,5,6,7,8,9。
  播放的效果为画面以正常速度的50倍进行快放，即1分钟的视频仅用1秒左右就播放完了，给人的感觉就是画面只显示了一瞬间，播放器闪退。
- **实际录像时长为1分钟，播放器进度条显示时长为16小时，且画面卡住不动。**

  SDK 接收的音视频帧时间戳单位是毫秒，造成这种情况是误将时间戳填成了微秒，使得1分钟的录像变为了16小时，画面其实并没有真正卡住，而是以相当于千分之一的速度慢放。
- **实际录像时长为1分钟，播放器进度条显示时长远超1分钟，且无画面或无声音或即无画面也无声音。**

  这种情况是音视频使用了不同的时间戳导致的，例如音频的时间戳从0开始计时，视频的时间戳从当前 UTC 事件开始计时，两者相差的时间非常大，使得播放器的进度条显示的时间异常以及播放异常。
  SDK 要求必须填写音视频帧的时间戳，但不对时间戳的参考时间做强制要求，用户可以根据自己的实际情况填写，例如长供电设备的时间戳可以使用精确到毫秒的 UTC 时间，断电设备的时间戳可以从0开始计时，或者也可以使用其他值同时作为音视频时间戳的基准。总之音频和视频一定要使用相同的参考时间或相同的时间源，不要各自用各自独立的时间源。

解决方法

保证音视频时间戳准确无误（手动计算时间戳时考虑误差、可变帧率等情况），保证音视频时间戳采用相同的参考时间或相同的时间源（例如编码器时钟、RTC 时钟、UTC 时钟、1毫秒 tick 时钟等）。
以上问题用户可以自行使用相关软件排查，软件详细使用方法见下文。

云存录像播放器异常

详细描述

云存录像播放异常，包括但不限于以下情况：
- 云存视频部分播放器能正常播放，其他播放器不能播放。
- 云存视频在线播放失败，下载后播放正常。
- 云存视频播放卡顿。
- 播放器闪退。

原因分析

目前常用的播放器功能差异较大，需要根据实际情况判断，常见原因有：

- 不支持软解码（或硬解码）H.264。
- 不支持软解码（或硬解码）H.265。
- 软解码 H.265 CPU 使用率过高导致卡顿（普通家用电脑（或手机）软解码高码率 4K H.265 视频可能非常吃力）。
- 云存视频异常，播放器解码错误导致闪退。

解决方法

建议根据播放器的实际情况进行调整，例如：

- 开启或关闭 H.264 软解码（硬解码）功能。
- 开启或关闭 H.265 软解码（硬解码）功能。
- 开启跳帧功能，优先保证画面流畅度。
- 开启纠错功能，跳过异常数据。

云存录像花屏

详细描述

云存录像回放时花屏（部分播放器可能会跳过花屏部分继续播放）。

原因分析

录像花屏说明云存上传过程中可能出现丢帧，例如由于网速原因导致云存缓存满，此时无法推送新的音视频帧进来，如果用户不做缓存则只能将这些帧丢掉，待网速恢复以后继续上传后续的音视频帧。这种情况下丢帧的地方就有可能异常，部分播放器会强制解码，部分播放器回跳过寻找下一个I帧继续播放。

解决方法

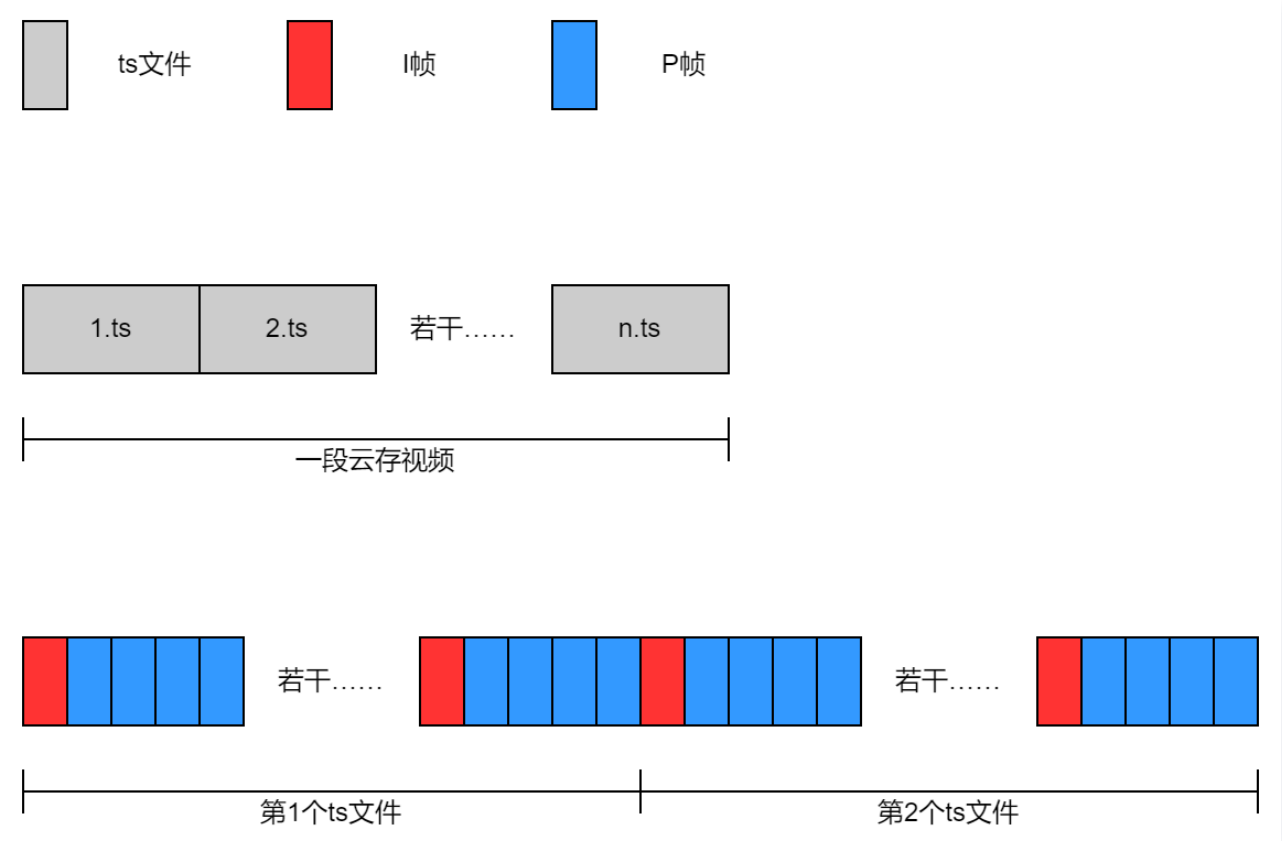
开启播放器的纠错功能等。
如果不希望花屏，用户可以将无法推送的视频数据暂存起来，等网络恢复以后继续发送；或丢弃无法推送的 P帧，直到下一个 I帧 再进行推送，以此减少花屏。

并行事件录像时长不正确

详细描述

并行事件录像时长不正确，例如第0秒触发了事件1，第15秒触发了事件2，第30秒同时结束了事件1和事件2。
拉取的事件列表中事件1时长30秒，事件1视频时长30秒，事件2时长15秒，视频时长20秒。

原因分析



如图所示，出于服务器负载等原因的考虑，目前 SDK 会将每10秒的视频分割为一个 ts 文件（或 ts 分片），ts 文件会在 I帧 处进行分割以免花屏，实际长度有一定波动。

> 例如：
> fps=20，GOP = 40帧，即每2秒一个 I帧，在第10秒时正好有一个 I帧，此时会分割一个 ts 文件。
> fps=20，GOP = 60帧，即每3秒一个 I帧，在第10秒时没有 I帧，因此会等到下一个 I帧，即第12秒处进行分割。

上述问题服务器中实际保存了【0秒至10秒】，【11秒至20秒】，【21秒至30秒】的三个视频文件，当查找事件2的视频即15秒 – 30秒的视频时，服务器会返回起止时间在15秒到30秒范围内的所有视频分片，即【11秒至20秒】，【21秒至30秒】这两个视频分片，因此实际看到的事件2的视频时长为20秒。同理，假设在第9秒触发了事件3，在第21秒结束事件3，事件3的实际持续时间为12秒，对应的视频为【0秒至10秒】，【11秒至20秒】，【21秒至30秒】这三个视频。

解决方法

该问题不影响云存录像，如果需要视频时长精确匹配事件时长，可以通过播放器精确定位视频时间来实现。

云存录像多1分钟

详细描述

云存录像实际长度为3分钟，回放时进度条显示为4分钟，且最后1分钟无法播放。

原因分析

全时云存顾名思义是需要持续录像的，正常的全时云存使用流程是初始化以后会收到 `iv_cs_push_stream_start_cb` 回调，之后用户应当持续推送音视频数据，直到因为云存套餐到期、退出云存等原因收到 `iv_cs_push_stream_stop_cb` 回调再停止推流。全时云存推流过程中可以调用 `iv_cs_event_start` 等接口触发事件。如果用户因特殊原因停止录像，SDK 内部会等待1分钟，如果1分钟内没有恢复就会结束录像，下次推流时恢复正常。等待的这1分钟内因为没有数据进来，导致回放时进度条显示的时间比实际录像时间长1分钟，且这1分钟无法播放。
开通了全时云存套餐却没有按 SDK 要求正确推流就会导致这种现象，问题中就是开通了全时套餐，但没有按 `iv_cs_push_stream_start_cb` 回调的指示开始推流，而是自行按照触发事件开始推流、结束事件停止推流的方式使用，SDK 发现无视频流继续等待1分钟后结束录像于是造成了问题中的现象。

解决方法

全时云存进行持续录像，或更换事件云存套餐。

事件云存无事件消息

详细描述

事件云存触发事件后只有视频，没有图片（或没有事件消息，或其他类似情形）。

原因分析

调用 `iv_cs_event_start` 等接口时没有检查错误码继续推流就可能导致上述问题。

解决方法

调用 `iv_cs_event_start` 等接口时请检查返回值，如有异常不建议继续推流，这种情况下 SDK 无法保证数据能够正常上传。

云存上传成功但播放器不播放

详细描述

通过设备端日志发现云存已经上传成功，回放时进度条显示有时间，但无法播放。

原因分析

云存视频要求必须上传音频和视频，如果用户只上传视频数据部分播放器可以正常播放，部分播放器则不行。
不能正常播放的常见原因是播放器优先使用音频帧的时间戳做音视频同步，因此播放器会一直向后读取，直到遇到音频数据才开始播放。
无音频，不播放（ts 里面标记有音频，实际没音频数据，播放器卡顿等）。

解决方法

- 正常上传音视频数据。
- 如果不需要音频（例如无麦克风、静音、保护隐私等）建议发送用0填充的音频数据帧。

- 如设备端无法上传音频，则需要修改播放器的相关设置，以 ffplay 为例，添加 `-an` 参数禁用音频，添加 `-sync video` 参数使用视频进行同步。

音视频传输和对讲相关问题

如何实现码率自适应

详细描述

实际网络环境波动较大，如何实现码率自适应。

原因分析

无

解决方法

- `iv_avt_init` 初始化参数 pstInitParm->congestion 中可以设置是否启用水位告警以及告警的有高中低三挡水位值，当 p2p 内部缓存的水位到达这个值的时候会收到 `iv_avt_notify_cb` 回调。
- 使用过程中主动调用 `iv_avt_get_send_stream_buf` 查询当前水位值。
- 使用过程中主动调用 `iv_avt_get_send_stream_status` 查询当前的瞬时网速和1秒内的平均网速。

用户根据以上3种方法的查询结果自行开发并实现码率自适应。

> **说明：**
> 下面给出一种实现思路，仅供参考。
> - 当发现 p2p 的水线超过一定值时，降低视频码率。例如当水位超过低水位时将视频码率降为原来的80%。网络正常的情况下 p2p 水位值很低，2mbps码率的视频水位值一般在100KB以下，该数值仅供参考，送入体积较大的 I帧、网络波动等都会影响水位值。
> - 推流过程中每间隔一定时间（例如1秒）调用 `iv_avt_get_send_stream_status` 获取获取网速信息。由于瞬时速度的波动较大，这里建议使用1秒内的平均传输速度，设置一定长度的队列（例如长度为5，如果调用间隔比较短可以适当加长窗口），将该数值存入队列同时删除队列内最旧的一个数值，去掉一个最高值去掉一个最低值，计算平均值。算出的平均值可用于控制码率，一般而言此数值与视频码率相近，当发现平均网速低于视频码率时主动降低视频码率到一个比平均网速更低的值。
>
> 用户可结合以上方法实现或借鉴 cubic 拥塞控制算法等的思想实现自己的码率自适应策略。对于 p2p 透传数据请参考 `iv_avt_p2p_set_buf_watermark`，`iv_avt_p2p_get_send_buf`，`iv_avt_p2p_get_send_status` 接口，具体实现思路类似。

发送音视频数据返回错误问题

详细描述

调用 `iv_avt_send_stream` 发送音视频数据返回错误，不同的错误码的原因不同。

原因分析

几种常见的错误码原因如下：

- 错误码为-303时，表示此时内部缓存满，送入数据失败，一般是网络原因导致此时的网络速度低于数据发送速度。
- 错误码为-305时，表示此时的 `visitor, channel, video_res_type` 三个参数中至少有一个值与 `iv_avt_start_real_play_cb` 通知的值不一致。
- 错误码为-306时，表示当前码率启动推流时送入的第一个视频帧不是 IDR 帧。
- 错误码为-308时，表示送入的音视频格式与 `iv_avt_get_av_enc_info_cb` 设置的格式不一致，或者本身送入的数据帧格式有问题，导致流媒体协议封装失败。

解决方法

- 出现错误码-303时，在这个错误之前一般都会有水位报警，需要降低码率，码率控制方法参考 如何实现码率自适应 。
- 出现错误码-305时，需要用户检查自己的代码参数配置是否存在问题。
- 出现错误码-306时，需要用户推送的第一个视频帧为 IDR 帧，也可以不用理会，等待编码器正常产生 IDR 帧，SDK 会把返回错误码的数据帧丢弃。
- 出现错误码-308时，首先需要用户检测 `iv_avt_get_av_enc_info_cb` 回调中设置的格式与实际数据帧是否匹配，如果匹配还出现该错误，需要用户将发生保存的数据保存下来，分析该数据的格式是否正确。

观看直播或者回放时画面卡顿或者花屏

详细描述

在小程序或者 App 端，观看设备的直播时，画面卡顿、不流畅或花屏。

原因分析

卡顿的原因有很多种，需要逐一排除，其排查方法如下：
将 App/小程序端收到的音视频流保存在本地，格式一般为 flv；使用第三方播放器（推荐 PotPlayer 或者 VLC）观看本地保存的音视频数据，如果仍然出现卡顿，则是从原因1开始分析，否则从原因2开始。

- 原因1：这种卡顿一般是因为视频数据有缺失，从本地保存的音视频流中提取出 H264/H265 裸数据（推荐使用 ffmpeg），使用 elecard 分析 H264/H265 裸数据，找到卡顿的时间点， 确认是否有丢帧（可根据 silce header 中的 frame_num 值判断）， 一般都是有丢帧的， 丢帧一般都是在设备端引起的。 在设备端查找该时间点发生的音视频数据， 调用 `iv_avt_send_stream` 时是否有错误， 或者编码器生成的数据是否有丢帧（可将编码器的 GOP 实时值打印出来判断）。
- 原因2：这种卡顿一般是网络带宽低于数据码率或者时序有问题引起的， 先判断卡顿点设备端是否有水位报警，然后使用 flv 分析工具分析数据的时序是否有问题， 如果时序没有问题则需要在设备端做码率控制。

解决方法

- 如果是设备端带宽原因引起的卡顿， 需要做码率自适应。

- 如果是设备端丢帧导致的， 则需要用户检查代码中的丢帧逻辑是否有问题。
- 如果是读取编码器数据有问题， 则需要用户检查业务逻辑中 CPU 是否占用太高或者取数据线程优先级太低。

观看直播或者回放时画面延时大或者黑屏

详细描述

在小程序或者 App 端， 观看设备的直播时， 画面延时大或者黑屏。

原因分析

- 画面延时大， 一般是音视频帧的PTS出现异常导致的， IoT Video SDK 要求送入的音视频帧PTS单位必须是毫秒， 如果配置的单位不是毫秒， 则会在观看时出现异常。
- 画面延时大， 还有可能是设备端缓存的音视频数据太多， 需要用户检查业务中缓存的数据是否太多。
- 画面延时大， 还可能是播放器缓存的音视频数据太多， 多半伴有音视频 PTS 的同步问题， 需要在 App 端或者小程序端检查音视频帧 PTS 中差值是否太大。
- 黑屏一般也是 PTS 出现问题， 最常见的是PTS出现了回环， 不是单调递增的， 导致播放出现问题。

解决办法

- 如果是 PTS 问题， 需要设备端用户检查送入的音视频帧 PTS 配置是否有问题， 设备端 IoT Video SDK 在发送时不会做对音视频帧做同步缓存或者修改其 PTS。
- 如果是设备端缓存数据多引起的问题， 则需要用户检查自己的业务逻辑问题， 设备端 IoT Video SDK 只会因为网络延迟大的原因缓存数据， 其他则不会缓存。
- 还有一种场景是， 有些摄像头设备带有 PTZ 功能， 在转动时为了滤除马达的声音， 不发送音频而是等待设备静止后发送， 这样也会破坏 PTS 的连续性， 造成延时很大， 推荐采用发送静音帧的方式来解决该问题。

物模型相关问题

字符串相关物模型上传失败

详细描述

字符串相关的物模型， 包括属性， 行为或者事件， 上传失败但又没有错误码返回。

原因分析

最常见的原因是物模型中字符串使用 json 封装或者传入其他特殊字符， 物模型的值最终会被封装到一个大 json 消息体中， 如果传入的字符串本身就是 json 或者有特殊字符， 破坏了整个消息体的 json 完整性， 上报的消息不合法会被后台丢弃。

解决办法

如果需要使用字符串传输 json 信息， 推荐使用 base64 编码后再传输。

其他问题

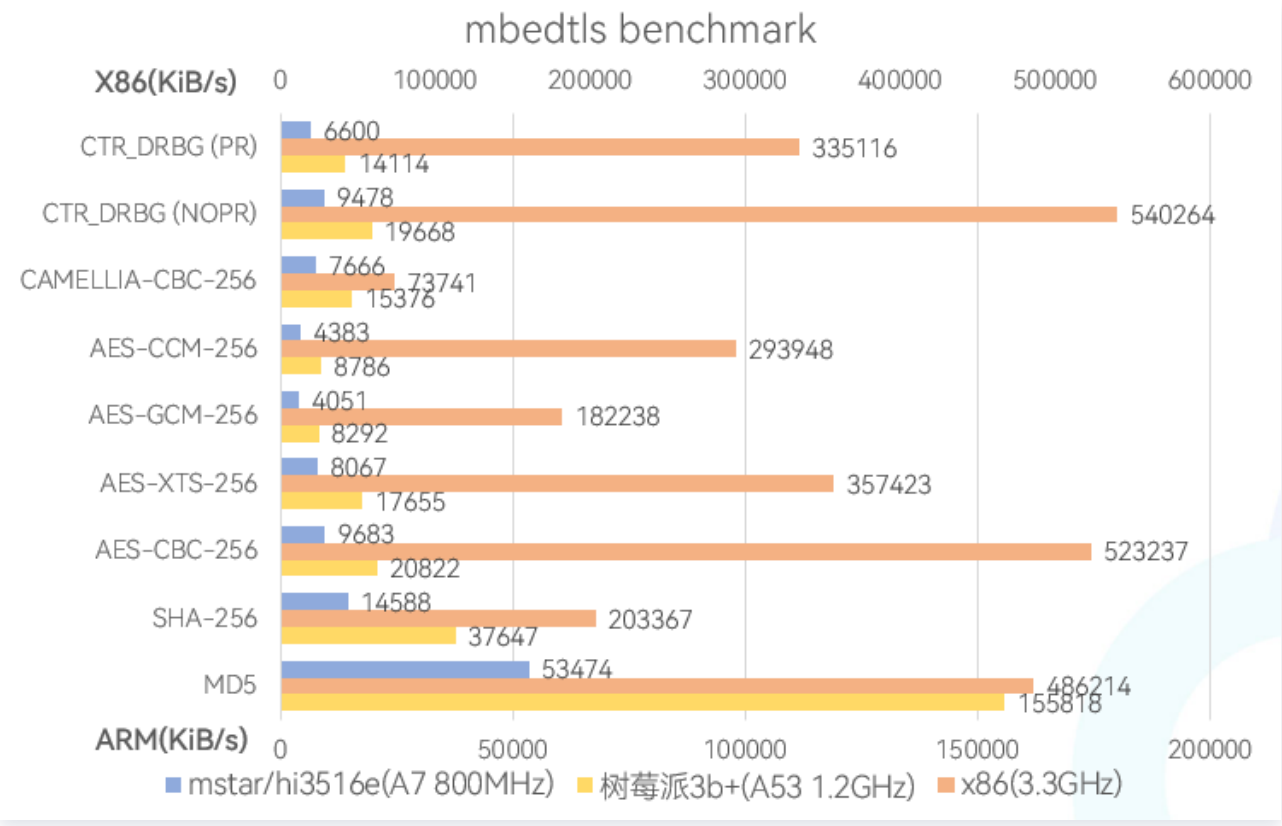
CPU 使用率高如何优化

详细描述

CPU 使用率高如何优化。

原因分析

一般都是由加密算法造成的，在低端芯片上更为明显。
云存和 P2P 视频传输默认都开启加密功能，云存和 P2P 视频传输目前采用的加密算法分别为 AES-CBC-128 和 AES-CTR-128。
下图给出部分加密算法在不同平台的跑分测试结果：



可以看到 AES-CBC-256 在 Hi3516E 系列的 CPU 上加密性能约为9600KB/s（AES-CBC-128 性能略高于 AES-CBC-256），假设云存视频的码率为2mbps，即每秒的数据量大约为256KB，计算可得 CPU 使用率约为3%，实际使用过程中受其他业务影响 CPU 使用率可能高于估算值。
P2P 视频传输采用的 AES-CTR-128 性能和 AES-CBC-128 性能相近，假设有多个用户同时向设备端拉流观看，CPU 的使用率会成倍增长，给设备端带来较大压力。
SDK 使用的 mbedtls 版本为2.16.9，用户可以自行下载对应版本并编译进行跑分测试，方法如下：
设置环境变量并编译

```
export CC="XXXXX"
export CFLAGS="-std=c99"
```

make
./programs/test/benchmark 即为性能测试程序，在设备上运行该程序查看跑分结果并估算 CPU 使用率。

解决方法

- 用户自行适配 mbedtls 的硬件加速相关接口，并替换 SDK 内默认的 mbdetls 库。
- 关闭加密功能（不推荐）。

常用工具

1. MediaInfo
   查看音视频文件的格式信息。
   下载地址
2. EasyIce
   分析 ts 视频文件或视频流。
   下载地址
3. Elecard StreamEye Tools
   分析 h264 视频文件。
   下载地址
4. Elecard HEVC Analyzer
   分析 h265 视频文件。
   下载地址
5. flvAnalyser
   Flv 分析工具。
   下载地址
6. VLC、PotPlayer、ffplay 等视频播放器。
7. mp4box
   mp4 文件分析工具。
   下载地址
8. Bento4
   mp4 文件分析工具。
   下载地址

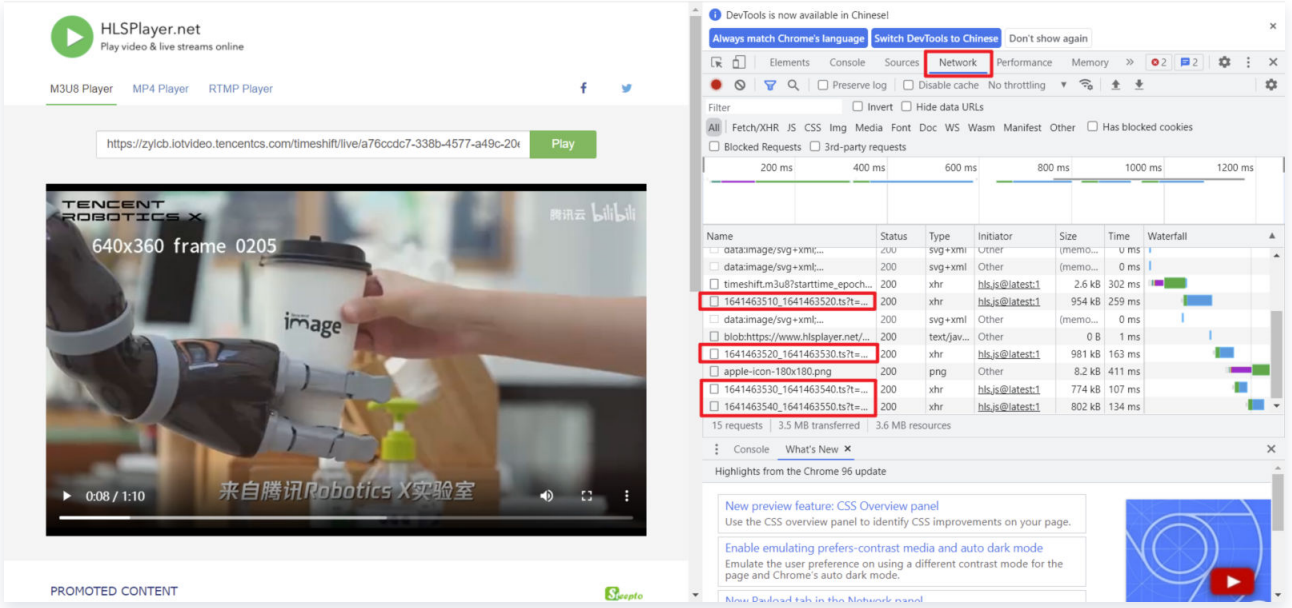
电脑端播放云存视频的方法

- 使用 VLC、PotPlayer 等播放器，选择播放在线视频，输入云存链接即可播放。
- 使用 ffplay，输入命令并替换云存链接 `ffplay <replace your link address>` ，加入 `-loglevel trace` 参数可显示详细信息，一定程度上有助于排查云存视频问题。
- 使用 Chrome 浏览器打开 `https://www.hlsplayer.net/` 等 m3u8 在线播放器，输入云存的播放地址并播放。

云存视频下载方法

方法一（推荐）

1. 使用 Chrome 浏览器打开 `https://www.hlsplayer.net/` 等 m3u8 在线播放器。
2. 按【F12】打开开发者工具，输入云存的播放地址并播放。
3. 如图所示，可以在 **Network** 标签页中看到若干 ts 文件。



4. 在这些 ts 文件上右键选择 **Copy > Copy link address**。
5. 将复制的链接放入任意下载器中进行下载（Chrome 直接访问该链接也可下载）。

方法二（推荐）

自行编写 Python 脚本进行下载，这里给出简易下载脚本，仅供参考。

```
from urllib.parse import urlparse
import requests

def get_m3u8(url):
    r = requests.get(url)
    if (r.status_code != 200):
        return None
    return r.content.decode("utf-8")

def make_ts_list(url, m3u8):
    ts_list = []
    m3u8_url = urlparse(url)
    url_head = m3u8_url.scheme + '://' + m3u8_url.hostname
    m3u8_lines = m3u8.split("\n")
```

```
    for each_line in m3u8_lines:
        if not each_line.startswith('#') and each_line != '':
            ts_list.append('%s%s' %(url_head, each_line))
    return ts_list

def download_ts(ts_list):
    for each_ts in ts_list:
        ts_url = urlparse(each_ts)
        pos = ts_url.path.rfind('/') + 1
        filename = ts_url.path[pos:]
        print("download " + filename)
        r = requests.get(each_ts)
        if (r.status_code != 200):
            return
        with open(filename, "wb") as fw:
            fw.write(r.content)
    print("download finish")

def main():
    aim_url =
"https://zylcb.iotvideo.tencentcs.com/timeshift/live/timeshift.m3u8"
    m3u8 = get_m3u8(aim_url)
    ts_list = make_ts_list(aim_url, m3u8)
    download_ts(ts_list)

if (__name__ == "__main__"):
    main()
```

方法三

1. Chrome 浏览器安装“网页资源嗅探器”等类似插件（此类插件众多，这里不做推荐）。
2. 打开 `https://www.hlsplayer.net/` 等 m3u8 在线播放器。
3. 输入云存的播放地址并播放，嗅探器会自动识别视频并下载。

注意事项

不要使用 ffmpeg，vlc 等工具进行下载，这类工具会进行二次封装或二次格式转换，导致原始信息丢失。

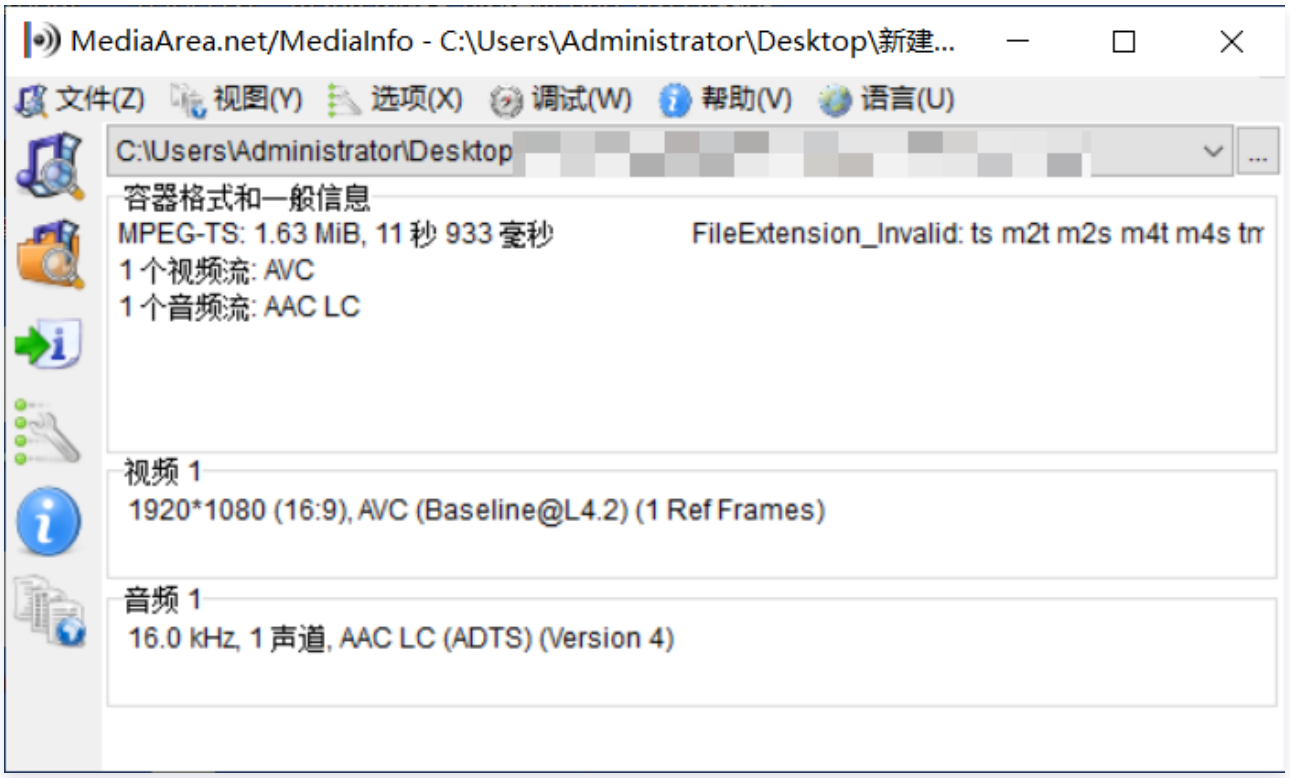
云存录像问题排查方法

准备工作：

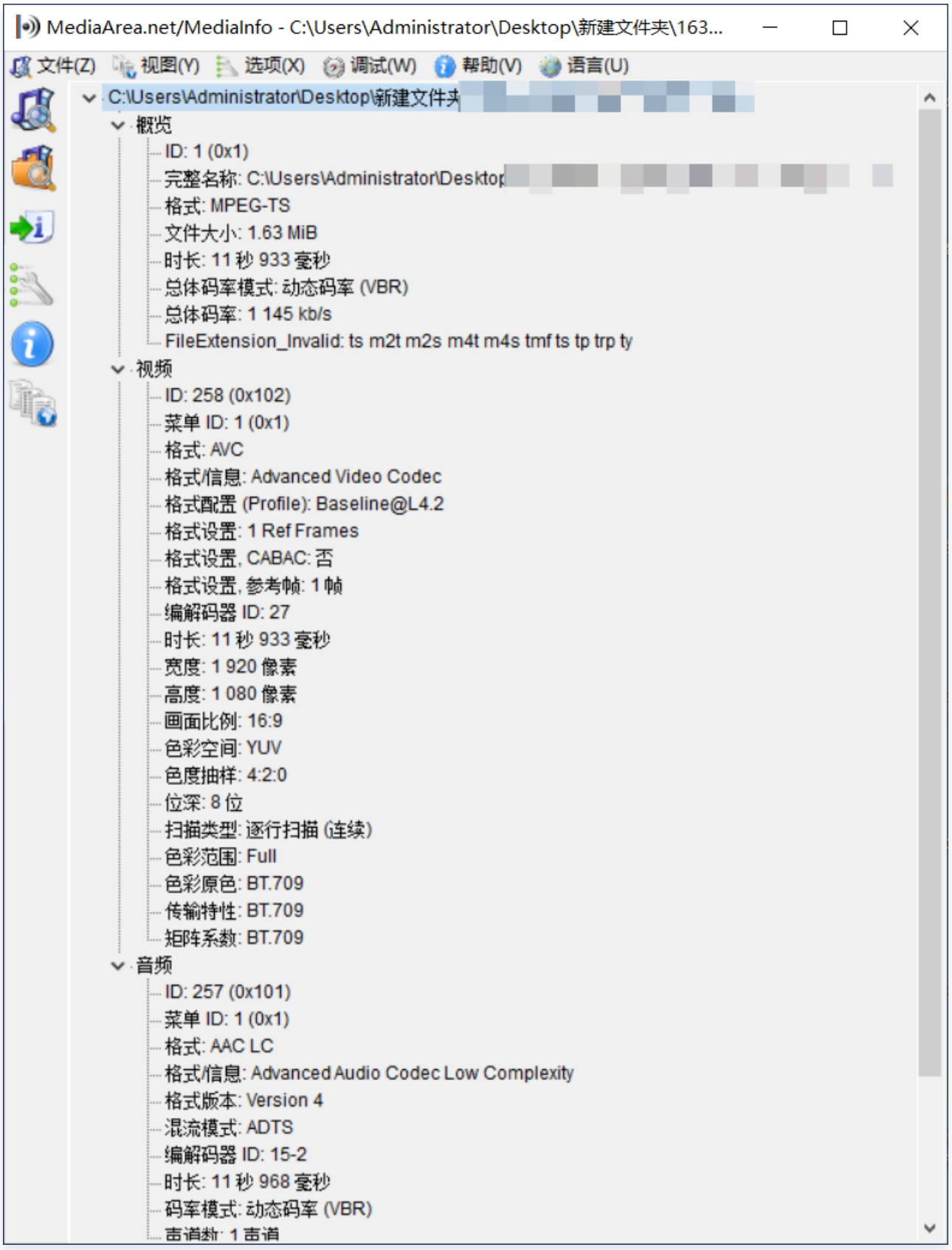
- 按前文所述方法下载云存视频。
- 准备相关工具软件。

云存录像基础检查

1. 使用 MediaInfo 打开视频文件即可看到基本信息。



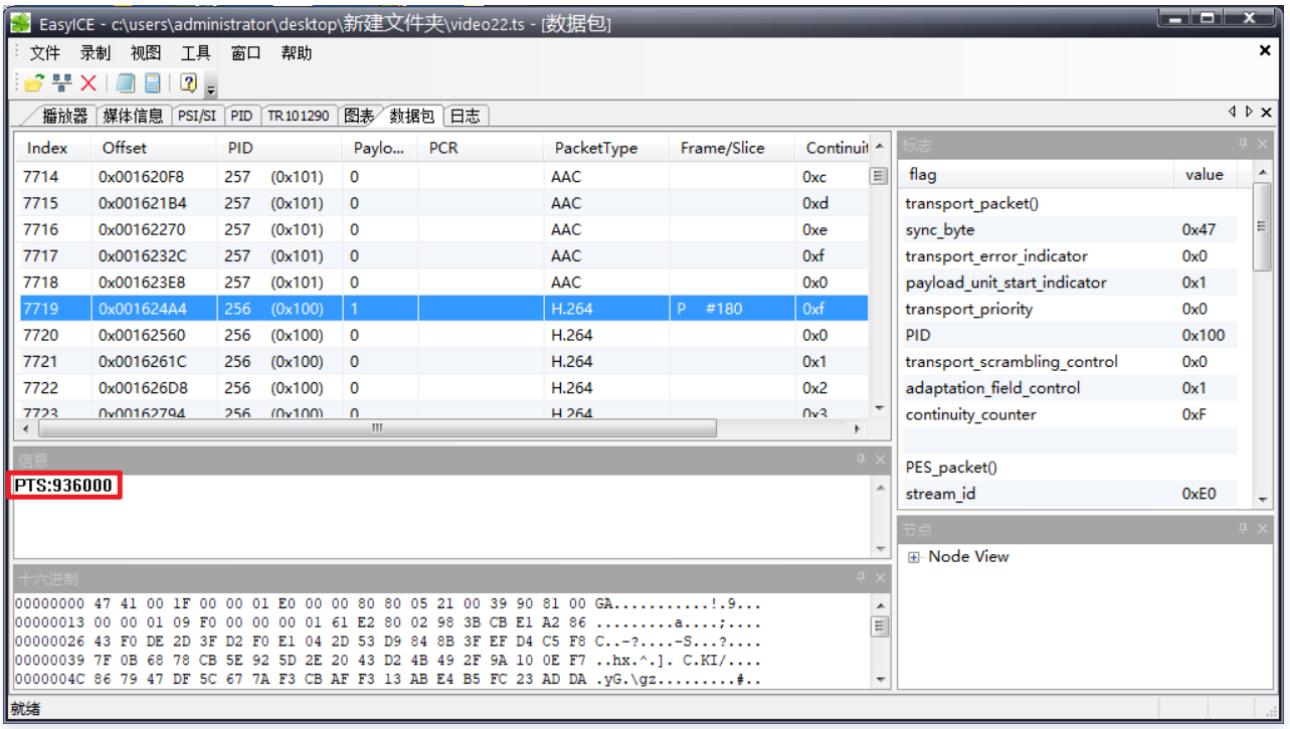
2. 单击**视图** > **树状图**可以看到更为详细的信息。



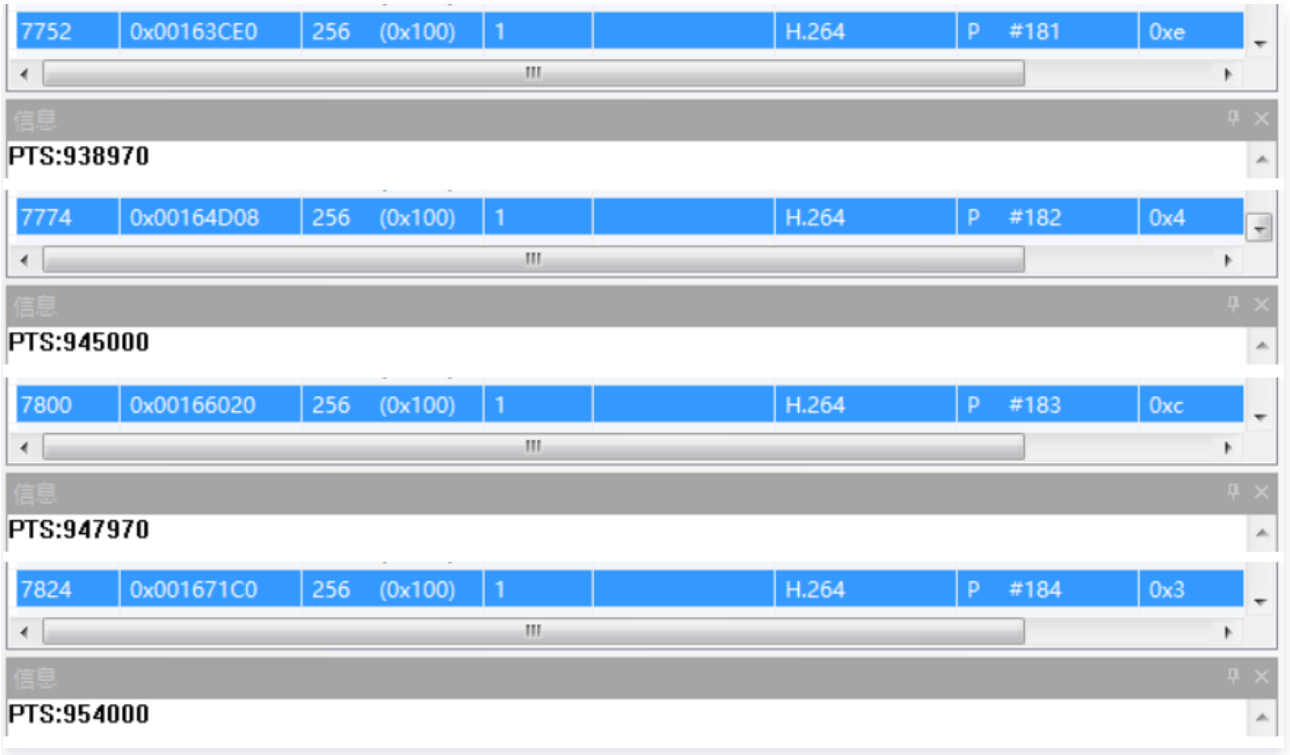
通过这些信息可以对视频做基本检查，例如视频分辨率是否正确、帧率是否正常、音频流数据是否缺失等。

云存录像时间异常问题

使用 EasyICE 打开视频，单击**数据包**标签页，找到任意几个连续的视频帧，计算它们的 pts 间隔是否正确。



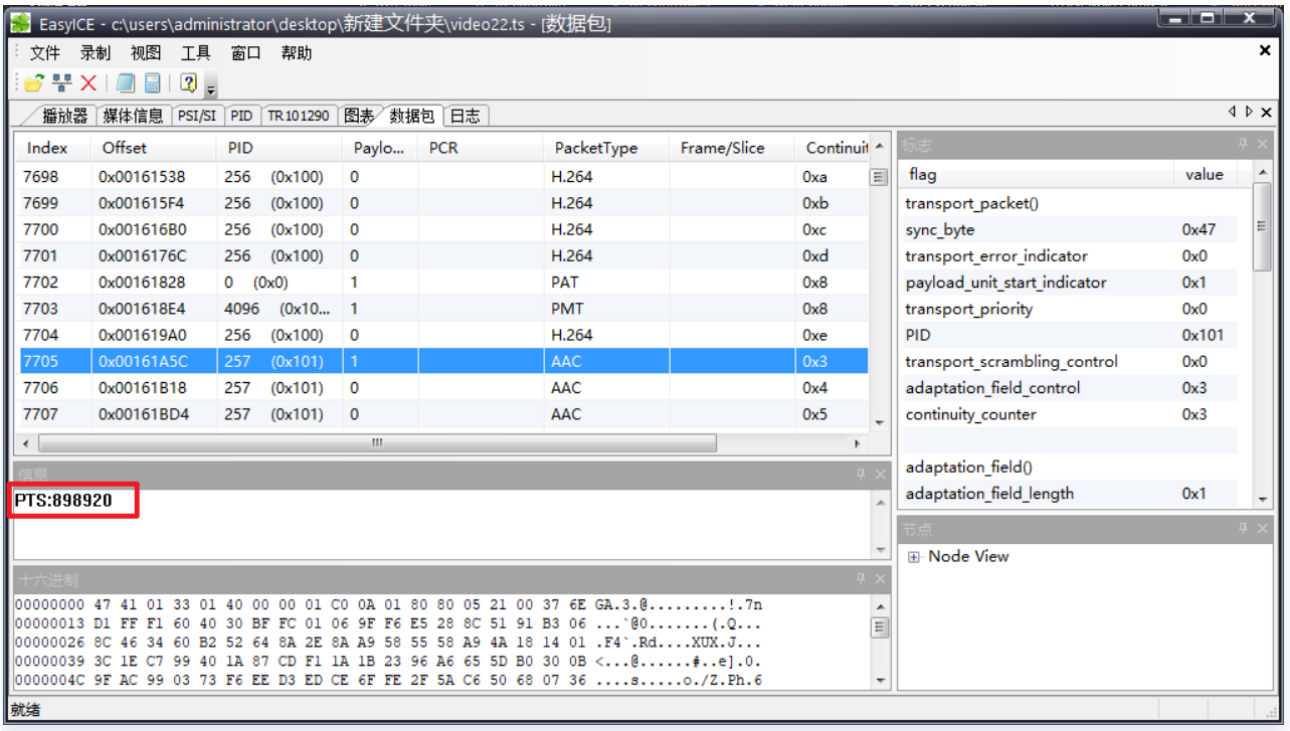
如下图所示，从第180帧开始几个视频帧的 PTS 分别为 936000，938970，945000，947970，954000。



MPEG-TS 标准规定音视频要使用一个90KHz的相对时钟或绝对时钟进行同步，因此将它们换算成毫秒需要除以90，计算结果分别为：10400，10433，10500，10533，10600。

假设设备端的帧率为25fps，这几个时间戳虽然分布不均匀，但之间的差值基本在50ms左右，可以认为视频时间戳正常。

同样地，音频帧也可以按照这种方式检查，但需要注意，云存会将所有音频非 aac 音频转换为 aac 格式，aac 每个音频帧为1024个采样点，假设音频采样率为44.1KHz，那么每个音频帧的时长为1024/44100=0.02322s=23.22ms。
接下来检查音视频时间戳是否同步，随意找几个视频帧附近的音频，观察音频帧和视频帧的 PTS 误差是否过大。

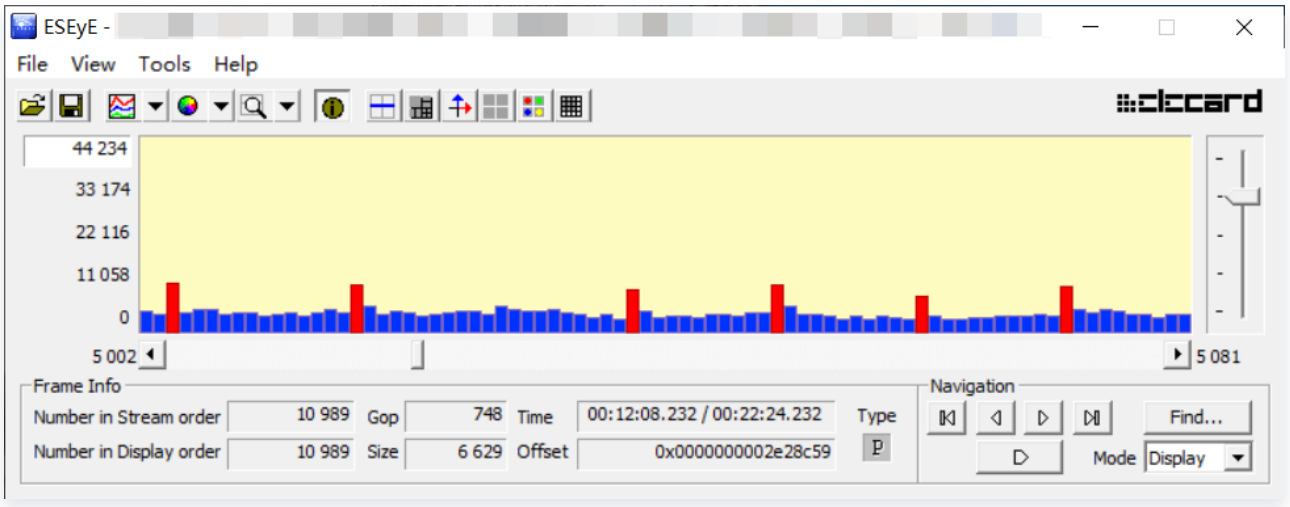


如上图所示，这是第180帧之前的一个音频帧，PTS 为 898920 即 9988ms，和视频帧 10400ms 相差412ms，这个误差有点大，但基本正常，一般来说误差在几百毫秒内都属于可接受范围。如果音视频之间的时间戳偏差过大，请在推流时检查时间戳是否正常。

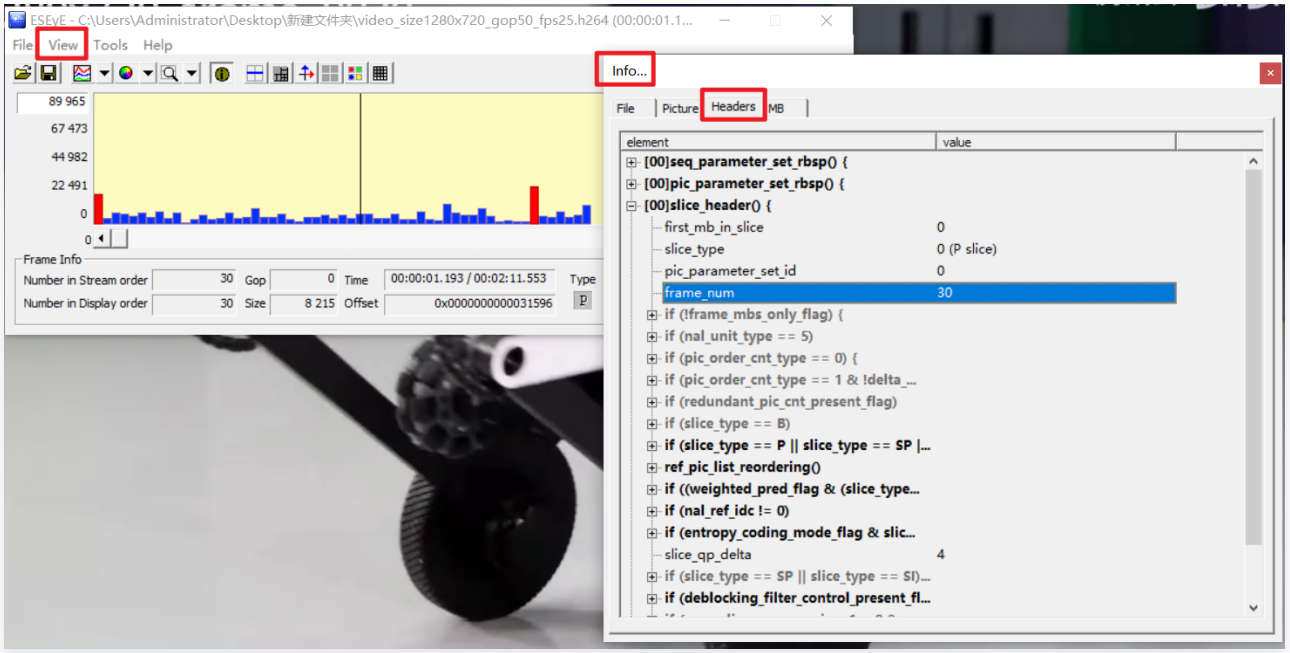
云存录像花屏问题

使用 Elecard StreamEye Tools 打开视频。

- 假设 GOP 为15帧，如图所示这里 I帧、P帧分布不均匀，明显有大量视频帧丢失，云存视频播放到这里就会花屏。



- 也可单击 View -> Info -> Headers -> slice_header() 查看前后两帧的 frame_num 是否连续，如果不连续则有丢帧。



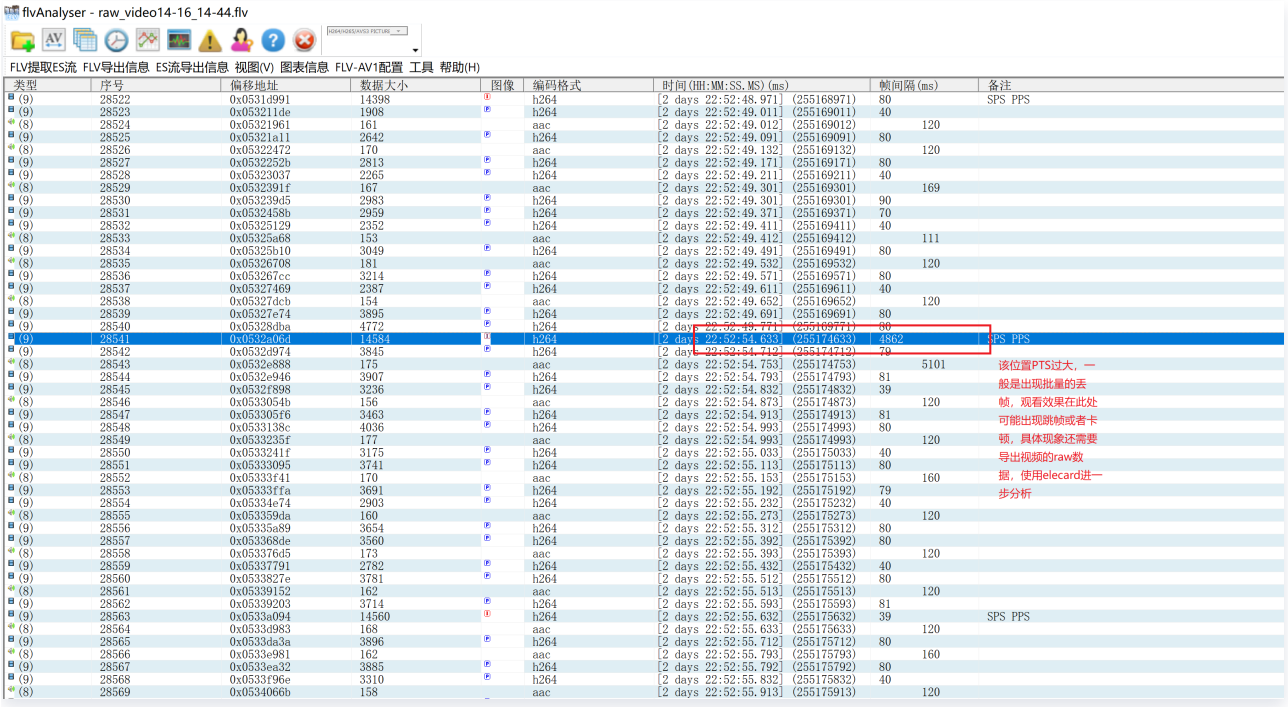
音视频传输和对讲问题排查方法

准备工作：

- App 端保存接收到的视频流。
- 准备相关工具软件。

FLV文件时序问题

使用 flvAnalyser 打开视频。
如果视频中有时序问题，则如下图所示：



类型	序号	偏移地址	数据大小	图像	编码格式	时间(HH:MM:SS.MS)(ms)	帧间隔(ms)	备注
(9)	28522	0x0531d991	14398		h264	[2 days 22:52:48.971] (255168971)	80	SPS PPS
(9)	28523	0x053211de	1908		h264	[2 days 22:52:49.011] (255169011)	40	
(8)	28524	0x05321961	161		aac	[2 days 22:52:49.012] (255169012)	120	
(9)	28525	0x05321a11	2642		h264	[2 days 22:52:49.091] (255169091)	80	
(8)	28526	0x05322472	170		aac	[2 days 22:52:49.132] (255169132)	120	
(9)	28527	0x0532252b	2813		h264	[2 days 22:52:49.171] (255169171)	80	
(9)	28528	0x05323037	2265		h264	[2 days 22:52:49.211] (255169211)	40	
(8)	28529	0x0532391f	167		aac	[2 days 22:52:49.301] (255169301)	169	
(9)	28530	0x053239d5	2983		h264	[2 days 22:52:49.301] (255169301)	90	
(9)	28531	0x0532458b	2959		h264	[2 days 22:52:49.371] (255169371)	70	
(9)	28532	0x05325129	2352		h264	[2 days 22:52:49.411] (255169411)	40	
(8)	28533	0x05325a68	153		aac	[2 days 22:52:49.412] (255169412)	111	
(9)	28534	0x05325b10	3049		h264	[2 days 22:52:49.491] (255169491)	80	
(8)	28535	0x05326708	181		aac	[2 days 22:52:49.532] (255169532)	120	
(9)	28536	0x053267cc	3214		h264	[2 days 22:52:49.571] (255169571)	80	
(9)	28537	0x05327469	2387		h264	[2 days 22:52:49.611] (255169611)	40	
(8)	28538	0x05327dcb	154		aac	[2 days 22:52:49.652] (255169652)	120	
(9)	28539	0x05327e74	3895		h264	[2 days 22:52:49.691] (255169691)	80	
(9)	28540	0x05328dba	4772		h264	[2 days 22:52:49.771] (255169771)	80	
(9)	28541	0x0532a06d	14584		h264	[2 days 22:52:54.633] (255174633)	4862	SPS PPS
(9)	28542	0x0532d974	3845		h264	[2 days 22:52:54.712] (255174712)	79	
(8)	28543	0x0532e888	175		aac	[2 days 22:52:54.753] (255174753)	5101	
(9)	28544	0x0532e946	3907		h264	[2 days 22:52:54.793] (255174793)	81	
(9)	28545	0x0532f898	3236		h264	[2 days 22:52:54.832] (255174832)	39	
(8)	28546	0x0533054b	156		aac	[2 days 22:52:54.873] (255174873)	120	
(9)	28547	0x053305f6	3463		h264	[2 days 22:52:54.913] (255174913)	81	
(9)	28548	0x0533138c	4036		h264	[2 days 22:52:54.993] (255174993)	80	
(8)	28549	0x0533235f	177		aac	[2 days 22:52:54.993] (255174993)	120	
(9)	28550	0x0533241f	3175		h264	[2 days 22:52:55.033] (255175033)	40	
(9)	28551	0x05333095	3741		h264	[2 days 22:52:55.113] (255175113)	80	
(8)	28552	0x05333f41	170		aac	[2 days 22:52:55.153] (255175153)	160	
(9)	28553	0x05333ffa	3691		h264	[2 days 22:52:55.192] (255175192)	79	
(9)	28554	0x05334e74	2903		h264	[2 days 22:52:55.232] (255175232)	40	
(8)	28555	0x053359da	160		aac	[2 days 22:52:55.273] (255175273)	120	
(9)	28556	0x05335a89	3654		h264	[2 days 22:52:55.312] (255175312)	80	
(9)	28557	0x053368de	3560		h264	[2 days 22:52:55.392] (255175392)	80	
(8)	28558	0x053376d5	173		aac	[2 days 22:52:55.393] (255175393)	120	
(9)	28559	0x05337791	2782		h264	[2 days 22:52:55.432] (255175432)	40	
(9)	28560	0x0533827e	3781		h264	[2 days 22:52:55.512] (255175512)	80	
(8)	28561	0x05339152	162		aac	[2 days 22:52:55.513] (255175513)	120	
(9)	28562	0x05339203	3714		h264	[2 days 22:52:55.593] (255175593)	81	
(9)	28563	0x0533a094	14560		h264	[2 days 22:52:55.632] (255175632)	39	SPS PPS
(8)	28564	0x0533d983	168		aac	[2 days 22:52:55.633] (255175633)	120	
(9)	28565	0x0533da3a	3896		h264	[2 days 22:52:55.712] (255175712)	80	
(8)	28566	0x0533e981	162		aac	[2 days 22:52:55.793] (255175793)	160	
(9)	28567	0x0533ea32	3885		h264	[2 days 22:52:55.792] (255175792)	80	
(9)	28568	0x0533f96e	3310		h264	[2 days 22:52:55.832] (255175832)	40	
(8)	28569	0x0534066b	158		aac	[2 days 22:52:55.913] (255175913)	120	